



eXpress

1.1.0

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

Generated by Doxygen 1.7.3

Wed May 23 2012 22:06:16



# Contents

<b>1</b>	<b>Class Index</b>	<b>1</b>
1.1	Class Hierarchy . . . . .	1
<b>2</b>	<b>Class Index</b>	<b>3</b>
2.1	Class List . . . . .	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
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	24
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	25
3.9.3.4	num_hits	25
3.9.3.5	sample_hit	25
3.10	FrequencyMatrix< T > Class Template Reference	25
3.10.1	Detailed Description	26
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	27
3.10.3.1	arr	27
3.10.3.2	increment	27
3.10.3.3	increment	27
3.10.3.4	operator()	28
3.10.3.5	operator()	28
3.10.3.6	set_logged	28
3.10.3.7	total	29
3.11	Indel Struct Reference	29
3.11.1	Detailed Description	29
3.11.2	Constructor & Destructor Documentation	30
3.11.2.1	Indel	30
3.11.3	Member Data Documentation	30
3.11.3.1	len	30
3.11.3.2	pos	30
3.12	Librarian Class Reference	30
3.12.1	Detailed Description	30
3.13	Library Struct Reference	30
3.13.1	Detailed Description	31
3.14	MapParser Class Reference	31
3.14.1	Detailed Description	31
3.14.2	Constructor & Destructor Documentation	32
3.14.2.1	MapParser	32
3.14.2.2	~MapParser	32
3.14.3	Member Function Documentation	32
3.14.3.1	reset_reader	32
3.14.3.2	targ_index	32
3.14.3.3	targ_lengths	33
3.14.3.4	threaded_parse	33
3.14.3.5	write_active	33

3.15	MarkovModel Class Reference	33
3.15.1	Detailed Description	34
3.16	MismatchTable Class Reference	34
3.16.1	Detailed Description	34
3.16.2	Constructor & Destructor Documentation	35
3.16.2.1	MismatchTable	35
3.16.3	Member Function Documentation	35
3.16.3.1	activate	35
3.16.3.2	append_output	35
3.16.3.3	log_likelihood	35
3.16.3.4	to_string	36
3.16.3.5	update	36
3.17	Parser Class Reference	36
3.17.1	Detailed Description	37
3.17.2	Member Function Documentation	37
3.17.2.1	header	37
3.17.2.2	next_fragment	37
3.17.2.3	reset	38
3.17.2.4	targ_index	38
3.17.2.5	targ_lengths	38
3.18	ParseThreadSafety Struct Reference	38
3.18.1	Detailed Description	39
3.19	RobertsFilter Class Reference	39
3.19.1	Detailed Description	39
3.20	RoundParams Struct Reference	39
3.20.1	Detailed Description	40
3.20.2	Constructor & Destructor Documentation	40
3.20.2.1	RoundParams	40
3.20.3	Member Data Documentation	40
3.20.3.1	mass	40
3.20.3.2	mass_var	40
3.20.3.3	tot_ambig_mass	41
3.20.3.4	var_sum	41
3.21	SAMParser Class Reference	41
3.21.1	Detailed Description	41
3.21.2	Constructor & Destructor Documentation	42
3.21.2.1	SAMParser	42
3.21.3	Member Function Documentation	42
3.21.3.1	header	42
3.21.3.2	next_fragment	42
3.21.3.3	targ_index	42
3.21.3.4	targ_lengths	43
3.22	SAMWriter Class Reference	43
3.22.1	Detailed Description	43
3.22.2	Constructor & Destructor Documentation	44
3.22.2.1	SAMWriter	44
3.22.2.2	~SAMWriter	44
3.22.3	Member Function Documentation	44
3.22.3.1	write_fragment	44
3.23	Sequence Class Reference	45

3.23.1	Detailed Description	45
3.23.2	Member Function Documentation	45
3.23.2.1	length	45
3.23.2.2	operator[]	46
3.24	SequenceFwd Class Reference	46
3.24.1	Detailed Description	47
3.24.2	Constructor & Destructor Documentation	47
3.24.2.1	SequenceFwd	47
3.24.2.2	SequenceFwd	47
3.24.2.3	SequenceFwd	47
3.24.2.4	~SequenceFwd	48
3.24.3	Member Function Documentation	48
3.24.3.1	length	48
3.24.3.2	operator=	48
3.24.3.3	operator[]	48
3.24.3.4	set	49
3.25	SequenceRev Class Reference	49
3.25.1	Detailed Description	49
3.25.2	Member Function Documentation	50
3.25.2.1	length	50
3.25.2.2	operator[]	50
3.26	SeqWeightTable Class Reference	50
3.26.1	Detailed Description	51
3.26.2	Constructor & Destructor Documentation	51
3.26.2.1	SeqWeightTable	51
3.26.3	Member Function Documentation	51
3.26.3.1	append_output	51
3.26.3.2	copy_expected	52
3.26.3.3	copy_observed	52
3.26.3.4	get_weight	52
3.26.3.5	increment_expected	52
3.26.3.6	increment_observed	53
3.26.3.7	normalize_expected	53
3.26.3.8	to_string	53
3.27	Target Class Reference	53
3.27.1	Detailed Description	54
3.27.2	Constructor & Destructor Documentation	55
3.27.2.1	Target	55
3.27.2.2	~Target	55
3.27.3	Member Function Documentation	55
3.27.3.1	add_mass	55
3.27.3.2	bundle	55
3.27.3.3	bundle	56
3.27.3.4	cached_effective_length	56
3.27.3.5	est_effective_length	56
3.27.3.6	id	56
3.27.3.7	incr_counts	57
3.27.3.8	length	57
3.27.3.9	lock	57
3.27.3.10	log_likelihood	57

3.27.3.11	mass	58
3.27.3.12	mass_var	58
3.27.3.13	name	58
3.27.3.14	rho	58
3.27.3.15	round_reset	58
3.27.3.16	seq	59
3.27.3.17	seq	59
3.27.3.18	solvable	59
3.27.3.19	solvable	59
3.27.3.20	tot_ambig_mass	59
3.27.3.21	tot_counts	60
3.27.3.22	uniq_counts	60
3.27.3.23	unlock	60
3.27.3.24	update_target_bias	60
3.27.3.25	var_sum	60
3.28	TargetTable Class Reference	61
3.28.1	Detailed Description	61
3.28.2	Constructor & Destructor Documentation	62
3.28.2.1	TargetTable	62
3.28.2.2	~TargetTable	62
3.28.3	Member Function Documentation	62
3.28.3.1	covar_size	62
3.28.3.2	get_covar	62
3.28.3.3	get_targ	63
3.28.3.4	merge_bundles	63
3.28.3.5	num_bundles	63
3.28.3.6	output_results	63
3.28.3.7	round_reset	64
3.28.3.8	size	64
3.28.3.9	threaded_bias_update	64
3.28.3.10	total_fpb	64
3.28.3.11	update_covar	64
3.28.3.12	update_total_fpb	65
3.29	ThreadSafeFragQueue Class Reference	65
3.29.1	Detailed Description	65
3.30	Writer Class Reference	65
3.30.1	Detailed Description	66
3.30.2	Member Function Documentation	66
3.30.2.1	write_fragment	66



# 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< T > . . . . .	25
Indel . . . . .	29
Librarian . . . . .	30
Library . . . . .	30
MapParser . . . . .	31
MarkovModel . . . . .	33
MismatchTable . . . . .	34
Parser . . . . .	36
BAMParser . . . . .	5
SAMPParser . . . . .	41
ParseThreadSafety . . . . .	38
RobertsFilter . . . . .	39
RoundParams . . . . .	39
Sequence . . . . .	45
SequenceFwd . . . . .	46
SequenceRev . . . . .	49
SeqWeightTable . . . . .	50
Target . . . . .	53
TargetTable . . . . .	61
ThreadSafeFragQueue . . . . .	65
Writer . . . . .	65
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< T >	25
Indel	29
Librarian	30
Library	30
MapParser	31
MarkovModel	33
MismatchTable	34
Parser	36
ParseThreadSafety	38
RobertsFilter	39
RoundParams	39
SAMParser	41
SAMWriter	43
Sequence	45
SequenceFwd	46
SequenceRev	49
SeqWeightTable	50
Target	53
TargetTable	61
ThreadSafeFragQueue	65

<a href="#">Writer</a> . . . . .	<a href="#">65</a>
----------------------------------	--------------------

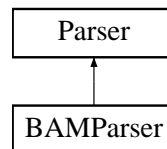
## Chapter 3

# 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 103 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 233 of file mapparser.cpp.

**3.1.2.2 BAMParser::~BAMParser ( ) [inline]**

[BAMParser](#) destructor deletes the reader

Definition at line 141 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 147 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 <a href="#">Fragment</a> to add mappings to
----------	---

**Returns**

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

Implements [Parser](#).

Definition at line 256 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 313 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 153 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 159 of file mapparser.h.

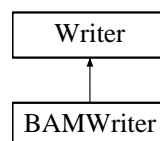
The documentation for this class was generated from the following files:

- 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 180 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 323 of file mapparser.cpp.

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

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

Definition at line 327 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 <a href="#">Fragment</a> to output
----------	--

Implements [Writer](#).

Definition at line 333 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 119 of file biascorrection.h.

### 3.3.2 Constructor & Destructor Documentation

#### 3.3.2.1 BiasBoss::BiasBoss ( double *alpha* )

[BiasBoss](#) Constructor

##### Parameters

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

Definition at line 142 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 195 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 171 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 216 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 159 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 177 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 <a href="#">Target</a> object in the bundle
<i>fmt</i>	a pointer to the (global) <a href="#">FragMassTable</a>

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](#) destructor. 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 <a href="#">Target</a> initially contained in the <a href="#">Bundle</a>
-------------	--

**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 <a href="#">Bundle</a> objects to merge
<i>b2</i>	a pointer to the other <a href="#">Bundle</a> 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](#) (TargID targ1, TargID targ2, double covar)
- double [get](#) (TargID targ1, TargID 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 ( TargID targ1, TargID 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 ( TargID targ1, TargID 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 <a href="#">FragHit</a> 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 Wed May 23 2012 22:06:16 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 <a href="#">FragHit</a> length
<i>mass</i>	a double for the mass (logged) of the observed <a href="#">FragHit</a>

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 81 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 100 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 <a href="#">FragHit</a> length to return the probability of
------------	--

**Returns**

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

Definition at line 75 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 105 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 95 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](#)
- `TargetID` [targ\\_id](#)
- `Target *` [mapped\\_targ](#)
- `SequenceFwd` [seq\\_l](#)
- `SequenceFwd` [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 63 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 141 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 153 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 124 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 134 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 119 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 129 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 94 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 114 of file fragments.h.

### 3.8.3.7 `Target* FragHit::mapped_targ`

a public pointer to the target mapped to

Definition at line 78 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 107 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 68 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 100 of file `fragments.h`.

### 3.8.3.11 `SequenceFwd FragHit::seq_l`

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

Definition at line 83 of file `fragments.h`.

### 3.8.3.12 `SequenceFwd FragHit::seq_r`

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

Definition at line 88 of file `fragments.h`.

### 3.8.3.13 `TargID FragHit::targ_id`

a public `TargID` for the target mapped to

Definition at line 73 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** ([Library](#) \*lib)
- [~Fragment](#) ()
- const [Library](#) \* **lib** ()
- 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
- void **mass** (double m)
- double **mass** () const
- void **sort\_hits** ()

### 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 180 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 18 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 <a href="#">FragHit</a> to be added
----------	---

Definition at line 31 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 248 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 236 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 242 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 probabalistic assignment

**Returns**

a randomly sampled [FragHit](#)

Definition at line 99 of file fragments.cpp.

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

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

**3.10 FrequencyMatrix< T > Class Template Reference**

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

**Public Member Functions**

- [FrequencyMatrix](#) ()
- [FrequencyMatrix](#) (size\_t m, size\_t n, T alpha, bool logged=true)
- T [operator\(\)](#) (size\_t i, size\_t j, bool normalized=true) const
- T [operator\(\)](#) (size\_t k) const
- void [increment](#) (size\_t i, size\_t j, T incr\_amt)

- void [increment](#) (size\_t k, T incr\_amt)
- double [arr](#) (size\_t k) const
- double [total](#) (size\_t i) const
- void [set\\_logged](#) (bool logged)
- void [fix](#) ()
- bool [is\\_fixed](#) () const
- size\_t [mode](#) (size\_t i) const

### 3.10.1 Detailed Description

`template<class T> class FrequencyMatrix< T >`

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 26 of file `frequencymatrix.h`.

### 3.10.2 Constructor & Destructor Documentation

**3.10.2.1** `template<class T> FrequencyMatrix< T >::FrequencyMatrix ( )`  
`[inline]`

dummy constructor

Definition at line 60 of file `frequencymatrix.h`.

**3.10.2.2** `template<class T> FrequencyMatrix< T >::FrequencyMatrix ( size_t m,`  
`size_t n, T 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 intial psuedo-counts (un-logged)
<i>logged</i>	bool that specifies if the table is in log space

Definition at line 132 of file frequencymatrix.h.

### 3.10.3 Member Function Documentation

**3.10.3.1** `template<class T> double FrequencyMatrix< T >::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 107 of file frequencymatrix.h.

**3.10.3.2** `template<class T> void FrequencyMatrix< T >::increment ( size_t i, size_t j, T 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 160 of file frequencymatrix.h.

**3.10.3.3** `template<class T> void FrequencyMatrix< T >::increment ( size_t k, T 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 181 of file frequencymatrix.h.

**3.10.3.4** `template<class T> T FrequencyMatrix< T >::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 154 of file frequencymatrix.h.

**3.10.3.5** `template<class T> T FrequencyMatrix< T >::operator() ( size_t i, size_t j, bool normalized = true ) 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)
<i>normalized</i>	a bool specifying whether or not the frequency should be normalized

**Returns**

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

Definition at line 142 of file frequencymatrix.h.

**3.10.3.6** `template<class T> void FrequencyMatrix< T >::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 187 of file frequencymatrix.h.

**3.10.3.7** `template<class T> double FrequencyMatrix< T >::total ( size_t i ) const`  
`[inline]`

a member function that returns the raw row sum

#### Parameters

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

#### Returns

a double specifying the raw row sum for the given distribution

Definition at line 114 of file `frequencymatrix.h`.

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

- `src/frequencymatrix.h`

## 3.11 Indel Struct Reference

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

#### Public Member Functions

- [Indel](#) (size\_t p, size\_t l)

#### Public Attributes

- size\_t [pos](#)
- size\_t [len](#)

### 3.11.1 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 39 of file `fragments.h`.

### 3.11.2 Constructor & Destructor Documentation

#### 3.11.2.1 `Indel::Indel ( size_t p, size_t / ) [inline]`

[Indel](#) constructor

Definition at line 54 of file fragments.h.

### 3.11.3 Member Data Documentation

#### 3.11.3.1 `size_t Indel::len`

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

Definition at line 49 of file fragments.h.

#### 3.11.3.2 `size_t Indel::pos`

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

Definition at line 44 of file fragments.h.

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

- `src/fragments.h`

## 3.12 Librarian Class Reference

### Public Member Functions

- **Librarian** (`size_t num_libs`)
- [Library](#) & **operator[]** (`size_t i`)
- const [Library](#) & **curr\_lib** () const
- void **set\_curr** (`size_t i`)
- `size_t size` () const

#### 3.12.1 Detailed Description

Definition at line 31 of file library.h.

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

- `src/library.h`

## 3.13 Library Struct Reference

```
#include <library.h>
```

### Public Attributes

- [MapParser](#) \* **map\_parser**
- [FLD](#) \* **fld**
- [MismatchTable](#) \* **mismatch\_table**
- [BiasBoss](#) \* **bias\_table**
- [TargetTable](#) \* **targ\_table**
- **size\_t n**
- **double mass\_n**

#### 3.13.1 Detailed Description

a struct for holding pointers to the global parameter tables (bias\_table, mismatch\_table, fld)

Definition at line 18 of file library.h.

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

- src/library.h

## 3.14 MapParser Class Reference

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

### Public Member Functions

- [MapParser](#) (std::string input\_file, std::string output\_file, [Library](#) \*lib, bool write\_active)
- [~MapParser](#) ()
- std::string & **in\_file\_name** ()
- void **threaded\_parse** ([ParseThreadSafety](#) \*thread\_safety, size\_t stop\_at=0)
- const TransIndex & **targ\_index** ()
- const TransIndex & **targ\_lengths** ()
- void **write\_active** (bool b)
- void **reset\_reader** ()

#### 3.14.1 Detailed Description

The [MapParser](#) 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 349 of file mapparser.h.

**3.14.2 Constructor & Destructor Documentation****3.14.2.1 MapParser::MapParser ( std::string *input\_file*, std::string *output\_file*, Library \* *lib*, bool *write\_active* )**

[MapParser](#) 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 98 of file mapparser.cpp.

**3.14.2.2 MapParser::~~MapParser ( )**

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

Definition at line 162 of file mapparser.cpp.

**3.14.3 Member Function Documentation****3.14.3.1 void MapParser::reset\_reader ( ) [inline]**

a member function that resets the input parser

Definition at line 418 of file mapparser.h.

**3.14.3.2 const TransIndex& MapParser::targ\_index ( ) [inline]**

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

**Returns**

the target-to-index map

Definition at line 400 of file mapparser.h.



**3.14.3.3** `const TransIndex& MapParser::targ_lengths ( ) [inline]`

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

**Returns**

the target-to-length map

Definition at line 406 of file mapparser.h.

**3.14.3.4** `void MapParser::threaded_parse ( ParseThreadSafety * thread_safety, size_t stop_at = 0 )`

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 queues with the processing thread
<i>stop_at</i>	a size_t indicating how many reads to process before stopping (disabled if 0)

Definition at line 169 of file mapparser.cpp.

**3.14.3.5** `void MapParser::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 413 of file mapparser.h.

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

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

**3.15 MarkovModel Class Reference****Public Member Functions**

- **MarkovModel** (size\_t order, size\_t window\_size, size\_t num\_pos, double alpha)

- 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** ()

### 3.15.1 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

## 3.16 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 p, double mass)
- void **fix** ()
- std::string **to\_string** () const
- void **append\_output** (std::ofstream &outfile) const

### 3.16.1 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 ride and to return likelihoods of mismatches in given reads. All values are stored and returned in log space. DOC

#### Author

Adam Roberts

#### Date

2011 Artistic License 2.0

Definition at line 24 of file mismatchmodel.h.

### 3.16.2 Constructor & Destructor Documentation

#### 3.16.2.1 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.

### 3.16.3 Member Function Documentation

#### 3.16.3.1 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.

#### 3.16.3.2 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 66 of file biascorrection.cpp.

#### 3.16.3.3 double MismatchTable::log\_likelihood ( const FragHit & *f* ) const

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

##### 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.

**3.16.3.4 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 345 of file mismatchmodel.cpp.

**3.16.3.5 void MismatchTable::update ( const FragHit & f, double p, 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>p</i>	the logged posterior probability of the alignment
<i>mass</i>	the logged mass of the fragment DOC

Definition at line 166 of file mismatchmodel.cpp.

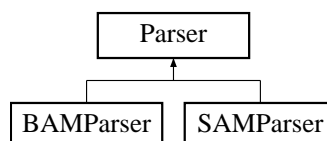
The documentation for this class was generated from the following files:

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

**3.17 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

### 3.17.1 Detailed Description

The [Parser](#) class is an abstract class that can be a [SAMParse](#)r or [BAMParse](#)r. 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 34 of file mapparser.h.

### 3.17.2 Member Function Documentation

#### 3.17.2.1 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 [BAMParse](#)r, and [SAMParse](#)r.

#### 3.17.2.2 virtual bool [Parser::next\\_fragment](#) ( [Fragment](#) &f ) [pure virtual]

a member function that loads all mappings of the next fragment

#### Parameters

<a href="#">f</a>	the empty <a href="#">Fragment</a> to add mappings to
-------------------	---

#### Returns

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

Implemented in [BAMParse](#)r, and [SAMParse](#)r.

**3.17.2.3** `virtual void Parser::reset ( ) [pure virtual]`

a member function that resets the parser and rewinds to the beginning of the input  
Implemented in [BAMParser](#).

**3.17.2.4** `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](#).

**3.17.2.5** `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 [SAMParser](#).

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

- `src/mapparser.h`

## 3.18 ParseThreadSafety Struct Reference

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

**Public Member Functions**

- **ParseThreadSafety** (size\_t q\_size)

**Public Attributes**

- [ThreadSafeFragQueue](#) **proc\_in**
- [ThreadSafeFragQueue](#) **proc\_on**
- [ThreadSafeFragQueue](#) **proc\_out**

### 3.18.1 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 40 of file threadsafety.h.

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

- src/threadsafety.h

## 3.19 RobertsFilter Class Reference

### Public Member Functions

- **RobertsFilter** (size\_t local\_size=DEFAULT\_LOC\_SIZE, size\_t global\_size=DEFAULT\_GLOB\_SIZE)
- bool **test\_and\_push** (const std::string &frag\_name)

### 3.19.1 Detailed Description

Definition at line 21 of file robertsfilter.h.

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

- src/robertsfilter.h
- src/robertsfilter.cpp

## 3.20 RoundParams Struct Reference

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

### Public Member Functions

- [RoundParams](#) ()

### Public Attributes

- double [mass](#)
- double [tot\\_ambig\\_mass](#)
- double [mass\\_var](#)
- double [var\\_sum](#)

### 3.20.1 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 35 of file targets.h.

### 3.20.2 Constructor & Destructor Documentation

#### 3.20.2.1 [RoundParams::RoundParams](#) ( ) `[inline]`

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

Definition at line 60 of file targets.h.

### 3.20.3 Member Data Documentation

#### 3.20.3.1 `double RoundParams::mass`

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

Definition at line 40 of file targets.h.

#### 3.20.3.2 `double RoundParams::mass_var`

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

Definition at line 50 of file targets.h.



### 3.20.3.3 double RoundParams::tot\_ambig\_mass

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

Definition at line 45 of file targets.h.

### 3.20.3.4 double RoundParams::var\_sum

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

Definition at line 55 of file targets.h.

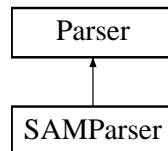
The documentation for this struct was generated from the following file:

- src/targets.h

## 3.21 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)

### 3.21.1 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 224 of file mapparser.h.

**3.21.2 Constructor & Destructor Documentation****3.21.2.1 SAMParser::SAMParser ( std::istream \* in )**

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

Definition at line 366 of file mapparser.cpp.

**3.21.3 Member Function Documentation****3.21.3.1 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 272 of file mapparser.h.

**3.21.3.2 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**

<i>f</i>	the empty <a href="#">Fragment</a> to add mappings to
----------	---

**Returns**

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

Implements [Parser](#).

Definition at line 422 of file mapparser.cpp.

**3.21.3.3 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 278 of file mapparser.h.

**3.21.3.4 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 284 of file mapparser.h.

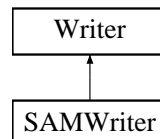
The documentation for this class was generated from the following files:

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

## 3.22 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)

### 3.22.1 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 302 of file mapparser.h.

**3.22.2 Constructor & Destructor Documentation****3.22.2.1 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 572 of file mapparser.cpp.

**3.22.2.2 SAMWriter::~~SAMWriter ( )**[SAMWriter](#) destructor flushes and deletes output stream

Definition at line 576 of file mapparser.cpp.

**3.22.3 Member Function Documentation****3.22.3.1 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 <a href="#">Fragment</a> to output
----------	--

Implements [Writer](#).

Definition at line 582 of file mapparser.cpp.

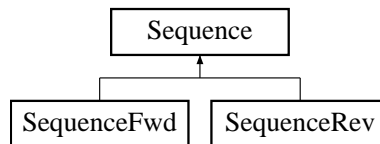
The documentation for this class was generated from the following files:

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

## 3.23 Sequence Class Reference

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

Inheritance diagram for Sequence:



### Public Member Functions

- virtual size\_t **operator[]** (const size\_t index) const =0
- virtual size\_t **get\_ref** (const size\_t index) const =0
- virtual void **update\_est** (const size\_t index, const size\_t nuc, float mass)=0
- virtual void **update\_obs** (const size\_t index, const size\_t nuc, float mass)=0
- virtual void **update\_exp** (const size\_t index, const size\_t nuc, float mass)=0
- virtual float **get\_prob** (const size\_t index, const size\_t nuc) const =0
- virtual float **get\_obs** (const size\_t index, const size\_t nuc) const =0
- virtual float **get\_exp** (const size\_t index, const size\_t nuc) const =0
- virtual bool **prob** () const =0
- virtual void **calc\_p\_vals** (std::vector< double > &p\_vals) const =0
- virtual size\_t **length** () const =0
- virtual bool **empty** () const =0

### 3.23.1 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 54 of file sequence.h.

### 3.23.2 Member Function Documentation

#### 3.23.2.1 virtual size\_t Sequence::length ( ) const [pure virtual]

a member function that returns the length of the encoded sequence

**Returns**

the length of the encoded sequence

Implemented in [SequenceFwd](#), and [SequenceRev](#).

**3.23.2.2** `virtual size_t Sequence::operator[] ( const size_t index ) const` [pure virtual]

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 < <i>_len</i> )
--------------	---

**Returns**

the encoded character at the given index

Implemented in [SequenceFwd](#), and [SequenceRev](#).

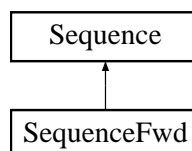
The documentation for this class was generated from the following file:

- `src/sequence.h`

**3.24 SequenceFwd Class Reference**

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

Inheritance diagram for SequenceFwd:

**Public Member Functions**

- [SequenceFwd](#) ()
- [SequenceFwd](#) (const std::string &seq, bool rev, bool prob=false)
- [SequenceFwd](#) (const [SequenceFwd](#) &other)
- [SequenceFwd](#) & [operator=](#) (const [SequenceFwd](#) &other)
- [~SequenceFwd](#) ()
- void [set](#) (const std::string &seq, bool rev)
- size\_t [operator\[\]](#) (const size\_t index) const
- size\_t [get\\_ref](#) (const size\_t index) const
- float [get\\_exp](#) (const size\_t index, const size\_t nuc) const

- float **get\_obs** (const size\_t index, const size\_t nuc) const
- void **update\_est** (const size\_t index, const size\_t nuc, float mass)
- void **update\_obs** (const size\_t index, const size\_t nuc, float mass)
- void **update\_exp** (const size\_t index, const size\_t nuc, float mass)
- float **get\_prob** (const size\_t index, const size\_t nuc) const
- bool **prob** () const
- bool **empty** () const
- size\_t **length** () const
- void **calc\_p\_vals** (std::vector< double > &p\_vals) const

### 3.24.1 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 89 of file sequence.h.

### 3.24.2 Constructor & Destructor Documentation

#### 3.24.2.1 SequenceFwd::SequenceFwd ( )

dummy constructor

Definition at line 16 of file sequence.cpp.

#### 3.24.2.2 SequenceFwd::SequenceFwd ( const std::string & seq, bool rev, bool prob = false )

[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 18 of file sequence.cpp.

#### 3.24.2.3 SequenceFwd::SequenceFwd ( const SequenceFwd & other )

[Sequence](#) copy constructor

**Parameters**

<i>other</i>	the <a href="#">Sequence</a> object to copy
--------------	---

Definition at line 29 of file sequence.cpp.

**3.24.2.4 SequenceFwd::~~SequenceFwd ( )**

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

Definition at line 55 of file sequence.cpp.

**3.24.3 Member Function Documentation****3.24.3.1 size\_t SequenceFwd::length ( ) const [inline, virtual]**

a member function that returns the length of the encoded sequence

**Returns**

the length of the encoded sequence

Implements [Sequence](#).

Definition at line 158 of file sequence.h.

**3.24.3.2 SequenceFwd & SequenceFwd::operator= ( const SequenceFwd & other )**

[Sequence](#) assignment constructor

**Parameters**

<i>other</i>	the <a href="#">Sequence</a> object to copy
--------------	---

Definition at line 39 of file sequence.cpp.

**3.24.3.3 size\_t SequenceFwd::operator[] ( const size\_t index ) const [virtual]**

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

Implements [Sequence](#).

Definition at line 80 of file sequence.cpp.



#### 3.24.3.4 void SequenceFwd::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

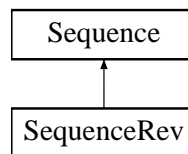
Definition at line 62 of file sequence.cpp.

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

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

## 3.25 SequenceRev Class Reference

Inheritance diagram for SequenceRev:



### Public Member Functions

- **SequenceRev** ([SequenceFwd](#) &seq)
- size\_t [length](#) () const
- bool **empty** () const
- void **set** (const std::string &seq, bool rev)
- size\_t [operator\[\]](#) (const size\_t index) const
- size\_t **get\_ref** (const size\_t index) const
- float **get\_obs** (const size\_t index, const size\_t nuc) const
- float **get\_exp** (const size\_t index, const size\_t nuc) const
- void **update\_est** (const size\_t index, const size\_t nuc, float mass)
- void **update\_obs** (const size\_t index, const size\_t nuc, float mass)
- void **update\_exp** (const size\_t index, const size\_t nuc, float mass)
- float **get\_prob** (const size\_t index, const size\_t nuc) const
- bool **prob** () const
- void **calc\_p\_vals** (std::vector< double > &p\_vals) const

### 3.25.1 Detailed Description

Definition at line 162 of file sequence.h.

### 3.25.2 Member Function Documentation

#### 3.25.2.1 `size_t SequenceRev::length ( ) const` `[inline, virtual]`

a member function that returns the length of the encoded sequence

##### Returns

the length of the encoded sequence

Implements [Sequence](#).

Definition at line 170 of file `sequence.h`.

#### 3.25.2.2 `size_t SequenceRev::operator[] ( const size_t index ) const` `[inline, virtual]`

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 < <code>_len</code> )
--------------	---

##### Returns

the encoded character at the given index

Implements [Sequence](#).

Definition at line 173 of file `sequence.h`.

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

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

## 3.26 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

### 3.26.1 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.

### 3.26.2 Constructor & Destructor Documentation

#### 3.26.2.1 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 28 of file biascorrection.cpp.

### 3.26.3 Member Function Documentation

#### 3.26.3.1 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
----------------	-----------------------

**3.26.3.2 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 <a href="#">SeqWeightTable</a> from which to copy the parameters
--------------	--

Definition at line 38 of file biascorrection.cpp.

**3.26.3.3 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 <a href="#">SeqWeightTable</a> from which to copy the parameters
--------------	--

Definition at line 33 of file biascorrection.cpp.

**3.26.3.4 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 59 of file biascorrection.cpp.

**3.26.3.5 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

Definition at line 43 of file biascorrection.cpp.

**3.26.3.6** `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 53 of file biascorrection.cpp.

**3.26.3.7** `void SeqWeightTable::normalize_expected ( )`

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

Definition at line 48 of file biascorrection.cpp.

**3.26.3.8** `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

## 3.27 Target Class Reference

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

#### Public Member Functions

- [Target](#) (const size\_t id, const std::string &name, const std::string &seq, bool prob\_seq, double alpha, const [Librarian](#) \*libs)

- [~Target](#) ()
- void [lock](#) ()
- void [unlock](#) ()
- const std::string & [name](#) () const
- TargID [id](#) () const
- const [Sequence](#) & [seq](#) (bool rev=false) const
- [Sequence](#) & [seq](#) (bool rev)
- 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 [solvable](#) ()
- void [solvable](#) (bool s)

### 3.27.1 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 77 of file targets.h.

### 3.27.2 Constructor & Destructor Documentation

**3.27.2.1** `Target::Target ( const size_t id, const std::string & name, const std::string & seq, bool prob_seq, double alpha, const Librarian * libs )`

[Target](#) Constructor

#### Parameters

<i>name</i>	a string that stores the target name
<i>seq</i>	a string that stores the target sequence
<i>prob_seq</i>	a bool that specifies if the sequence is to be treated probabilistically (for RDD detection)
<i>alpha</i>	a double that specifies the initial 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 24 of file targets.cpp.

**3.27.2.2** `Target::~~Target ( ) [inline]`

[Target](#) Destructor deletes bias vectors

Definition at line 187 of file targets.h.

### 3.27.3 Member Function Documentation

**3.27.3.1** `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 51 of file targets.cpp.

**3.27.3.2** `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 287 of file targets.h.

### 3.27.3.3 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 <a href="#">Bundle</a> to set this <a href="#">Target</a> as a member of
----------	---

Definition at line 293 of file targets.h.

### 3.27.3.4 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 152 of file targets.cpp.

### 3.27.3.5 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 <a href="#">FLD</a> 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 130 of file targets.cpp.

### 3.27.3.6 TargID Target::id ( ) const [inline]

a member function that returns the target id

#### Returns

TargID target ID



Definition at line 215 of file targets.h.

**3.27.3.7** `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 308 of file targets.h.

**3.27.3.8** `size_t Target::length ( ) const [inline]`

a member function that returns the target length

#### Returns

target length

Definition at line 233 of file targets.h.

**3.27.3.9** `void Target::lock ( ) [inline]`

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

Definition at line 198 of file targets.h.

**3.27.3.10** `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 <a href="#">FragHit</a> to return the likelihood of being originated from this target
<i>with_pseudo</i>	a <a href="#">FragHit</a> 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 97 of file targets.cpp.

**3.27.3.11 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 83 of file targets.cpp.

**3.27.3.12 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 90 of file targets.cpp.

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

a member function that returns the target name

**Returns**

string containing target name

Definition at line 209 of file targets.h.

**3.27.3.14 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 74 of file targets.cpp.

**3.27.3.15 void Target::round\_reset ( )**

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

Definition at line 67 of file targets.cpp.

**3.27.3.16** `const Sequence& Target::seq ( bool rev = false ) const` `[inline]`

a member function that returns the target sequence (const)

#### Returns

string containing target sequence

Definition at line 221 of file targets.h.

**3.27.3.17** `Sequence& Target::seq ( bool rev )` `[inline]`

a member function that returns the target sequence (non-const)

#### Returns

string containing target sequence

Definition at line 227 of file targets.h.

**3.27.3.18** `bool Target::solvable ( )` `[inline]`

a member function that returns the `_solvable` flag

#### Returns

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

Definition at line 350 of file targets.h.

**3.27.3.19** `void Target::solvable ( bool s )` `[inline]`

a member function that sets the `_solvable` flag

#### Parameters

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

Definition at line 356 of file targets.h.

**3.27.3.20** `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 264 of file targets.h.

### 3.27.3.21 `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 275 of file targets.h.

### 3.27.3.22 `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 281 of file targets.h.

### 3.27.3.23 `void Target::unlock ( )` `[inline]`

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

Definition at line 203 of file targets.h.

### 3.27.3.24 `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 curent parameters

#### Parameters

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

Definition at line 160 of file targets.cpp.

### 3.27.3.25 `double Target::var_sum ( ) const` `[inline]`

a member function that returns the (logged) weighted sum of the variance on the as-signments

**Returns**

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

Definition at line 258 of file targets.h.

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

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

## 3.28 TargetTable Class Reference

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

**Public Member Functions**

- [TargetTable](#) (const std::string &targ\_fasta\_file, bool prob\_seqs, double alpha, const AlphaMap \*alpha\_map, const [Librarian](#) \*libs)
- [~TargetTable](#) ()
- [Target \\* get\\_targ](#) (TargID id)
- void [round\\_reset](#) ()
- size\_t [size](#) () const
- double [total\\_fpb](#) () const
- void [update\\_total\\_fpb](#) (double incr\_amt)
- void [update\\_covar](#) (TargID targ1, TargID targ2, double covar)
- double [get\\_covar](#) (TargID targ1, TargID 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=false, bool output\_edits=false)
- void [threaded\\_bias\\_update](#) (boost::mutex \*mut)

### 3.28.1 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 373 of file targets.h.

### 3.28.2 Constructor & Destructor Documentation

**3.28.2.1** `TargetTable::TargetTable ( const std::string & targ_fasta_file, bool prob_seqs, double alpha, const AlphaMap * alpha_map, const Librarian * libs )`

[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>prob_seqs</i>	a bool that specifies if the sequence is to be treated probabilistically (for RDD detection)
<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 180 of file targets.cpp.

**3.28.2.2** `TargetTable::~~TargetTable ( )`

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

Definition at line 274 of file targets.cpp.

### 3.28.3 Member Function Documentation

**3.28.3.1** `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 488 of file targets.h.

**3.28.3.2** `double TargetTable::get_covar ( TargID targ1, TargID 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 482 of file targets.h.

**3.28.3.3 Target \* TargetTable::get\_targ ( TargetID *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 305 of file targets.cpp.

**3.28.3.4 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 <a href="#">Bundle</a> to merge
<i>b2</i>	a pointer to the second <a href="#">Bundle</a> to merge

**Returns**

a pointer to the merged [Bundle](#)

Definition at line 310 of file targets.cpp.

**3.28.3.5 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 319 of file targets.cpp.

**3.28.3.6 void TargetTable::output\_results ( std::string *output\_dir*, size\_t *tot\_counts*, bool *output\_varcov* = false, bool *output\_edits* = false )**

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
<i>output_-varcov</i>	boolean specifying whether to also output the editing p-values

Definition at line 385 of file targets.cpp.

**3.28.3.7 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 324 of file targets.cpp.

**3.28.3.8 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 452 of file targets.h.

**3.28.3.9 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 578 of file targets.cpp.

**3.28.3.10 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 566 of file targets.cpp.

**3.28.3.11 void TargetTable::update\_covar ( TargID targ1, TargID 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 473 of file targets.h.

**3.28.3.12 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 572 of file targets.cpp.

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

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

**3.29 ThreadSafeFragQueue Class Reference****Public Member Functions**

- **ThreadSafeFragQueue** (size\_t max\_size)
- **Fragment \* pop** (bool block=true)
- void **push** (**Fragment** \*frag)
- bool **empty** (bool block=false)

**3.29.1 Detailed Description**

Definition at line 19 of file threadsafety.h.

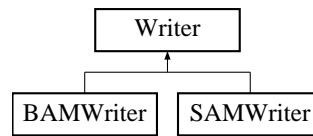
The documentation for this class was generated from the following files:

- src/threadsafety.h
- src/threadsafety.cpp

**3.30 Writer Class Reference**

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

Inheritance diagram for Writer:



## Public Member Functions

- virtual void `write_fragment (Fragment &f)=0`

### 3.30.1 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 77 of file `mapparser.h`.

### 3.30.2 Member Function Documentation

#### 3.30.2.1 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

<i>f</i>	the processed <code>Fragment</code> 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](#)
- ~MapParser
  - MapParser, [32](#)
- ~SAMWriter
  - SAMWriter, [44](#)
- ~SequenceFwd
  - SequenceFwd, [48](#)
- ~Target
  - Target, [55](#)
- ~TargetTable
  - TargetTable, [62](#)
- activate
  - MismatchTable, [35](#)
- add\_map\_end
  - Fragment, [24](#)
- add\_mass
  - Target, [55](#)
- add\_val
  - FLD, [18](#)
- append\_output
  - BiasBoss, [10](#)
  - FLD, [18](#)
  - MismatchTable, [35](#)
  - SeqWeightTable, [51](#)
- arr
  - FrequencyMatrix, [27](#)
- 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, [55](#), [56](#)
- bundles
  - BundleTable, [14](#)
- BundleTable, [13](#)
  - ~BundleTable, [14](#)
  - bundles, [14](#)
  - BundleTable, [14](#)
  - create\_bundle, [14](#)
  - merge, [15](#)
  - size, [15](#)
- cached\_effective\_length
  - Target, [56](#)
- cdf
  - FLD, [18](#)
- copy\_expected
  - SeqWeightTable, [51](#)
- copy\_observed
  - SeqWeightTable, [52](#)
- counts

- Bundle, 12
- covar\_size
  - TargetTable, 62
- CovarTable, 15
  - get, 16
  - increment, 16
  - size, 16
- create\_bundle
  - BundleTable, 14
- deletes\_l
  - FragHit, 21
- deletes\_r
  - FragHit, 21
- est\_effective\_length
  - Target, 56
- 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
- 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, 25
  - sample\_hit, 25
- FrequencyMatrix, 25
  - arr, 27
  - FrequencyMatrix, 26
  - increment, 27
  - operator(), 27, 28
  - set\_logged, 28
  - total, 28
- get
  - CovarTable, 16
- get\_covar
  - TargetTable, 62
- get\_targ
  - TargetTable, 63
- get\_target\_bias
  - BiasBoss, 10
- get\_weight
  - SeqWeightTable, 52
- header
  - BAMParser, 6
  - Parser, 37
  - SAMParser, 42
- hits
  - Fragment, 24
- id
  - Target, 56
- incr\_counts
  - Bundle, 12
  - Target, 57
- increment
  - CovarTable, 16
  - FrequencyMatrix, 27
- increment\_expected
  - SeqWeightTable, 52
- increment\_observed
  - SeqWeightTable, 53
- Indel, 29
  - Indel, 30
  - 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, 45
  - SequenceFwd, 48
  - SequenceRev, 50
  - Target, 57
- Librarian, 30
- Library, 30
- lock
  - Target, 57
- log\_likelihood
  - MismatchTable, 35
  - Target, 57
- MapParser, 31
  - ~MapParser, 32
  - MapParser, 32
  - reset\_reader, 32
  - targ\_index, 32
  - targ\_lengths, 32
  - threaded\_parse, 33
  - write\_active, 33
- mapped\_targ
  - FragHit, 22
- MarkovModel, 33
- mass
  - RoundParams, 40
  - Target, 57
- mass\_var
  - RoundParams, 40
  - Target, 58
- mate\_1
  - FragHit, 22
- max\_val
  - FLD, 18
- mean
  - FLD, 18
- merge
  - BundleTable, 15
- merge\_bundles
  - TargetTable, 63
- min\_val
  - FLD, 19
- MismatchTable, 34
  - activate, 35
  - append\_output, 35
  - log\_likelihood, 35
  - MismatchTable, 35
  - to\_string, 36
  - update, 36
- name
  - FragHit, 22
  - Fragment, 24
  - Target, 58
- next\_fragment
  - BAMParser, 6
  - Parser, 37
  - SAMParser, 42
- normalize\_expectations
  - BiasBoss, 10
- normalize\_expected
  - SeqWeightTable, 53
- num\_bundles
  - TargetTable, 63
- num\_hits
  - Fragment, 25
- operator()
  - FrequencyMatrix, 27, 28
- operator=
  - SequenceFwd, 48
- output\_results
  - TargetTable, 63
- pair\_status
  - FragHit, 21
- Parser, 36
  - header, 37
  - next\_fragment, 37
  - reset, 37
  - targ\_index, 38
  - targ\_lengths, 38
- ParseThreadSafety, 38
- pdf
  - FLD, 19
- pos
  - Indel, 30
- reset
  - BAMParser, 6
  - Parser, 37
- reset\_reader
  - MapParser, 32
- rho

- Target, 58
- right
  - FragHit, 22
- RobertsFilter, 39
- round\_reset
  - Target, 58
  - TargetTable, 64
- RoundParams, 39
  - mass, 40
  - mass\_var, 40
  - RoundParams, 40
  - tot\_ambig\_mass, 40
  - var\_sum, 41
- SAMParser, 41
  - header, 42
  - next\_fragment, 42
  - SAMParser, 42
  - targ\_index, 42
  - targ\_lengths, 43
- sample\_hit
  - Fragment, 25
- SAMWriter, 43
  - ~SAMWriter, 44
  - SAMWriter, 44
  - write\_fragment, 44
- seq
  - Target, 58, 59
- seq\_l
  - FragHit, 23
- seq\_r
  - FragHit, 23
- Sequence, 45
  - length, 45
- SequenceFwd, 46
  - ~SequenceFwd, 48
  - length, 48
  - operator=, 48
  - SequenceFwd, 47
  - set, 48
- SequenceRev, 49
  - length, 50
- SeqWeightTable, 50
  - append\_output, 51
  - copy\_expected, 51
  - copy\_observed, 52
  - get\_weight, 52
  - increment\_expected, 52
  - increment\_observed, 53
  - normalize\_expected, 53
  - SeqWeightTable, 51
  - to\_string, 53
- set
  - SequenceFwd, 48
- set\_logged
  - FrequencyMatrix, 28
- size
  - Bundle, 13
  - BundleTable, 15
  - CovarTable, 16
  - TargetTable, 64
- solvable
  - Target, 59
- targ\_id
  - FragHit, 23
- targ\_index
  - BAMParser, 7
  - MapParser, 32
  - Parser, 38
  - SAMParser, 42
- targ\_lengths
  - BAMParser, 7
  - MapParser, 32
  - Parser, 38
  - SAMParser, 43
- Target, 53
  - ~Target, 55
  - add\_mass, 55
  - bundle, 55, 56
  - cached\_effective\_length, 56
  - est\_effective\_length, 56
  - id, 56
  - incr\_counts, 57
  - length, 57
  - lock, 57
  - log\_likelihood, 57
  - mass, 57
  - mass\_var, 58
  - name, 58
  - rho, 58
  - round\_reset, 58
  - seq, 58, 59
  - solvable, 59
  - Target, 55
  - tot\_ambig\_mass, 59
  - tot\_counts, 60
  - uniq\_counts, 60
  - unlock, 60
  - update\_target\_bias, 60

- var\_sum, 60
- targets
  - Bundle, 13
- TargetTable, 61
  - ~TargetTable, 62
  - covar\_size, 62
  - get\_covar, 62
  - get\_targ, 63
  - merge\_bundles, 63
  - num\_bundles, 63
  - output\_results, 63
  - round\_reset, 64
  - size, 64
  - TargetTable, 62
  - threaded\_bias\_update, 64
  - total\_fpb, 64
  - update\_covar, 64
  - update\_total\_fpb, 65
- threaded\_bias\_update
  - TargetTable, 64
- threaded\_parse
  - MapParser, 33
- ThreadSafeFragQueue, 65
- to\_string
  - BiasBoss, 10
  - FLD, 19
  - MismatchTable, 36
  - SeqWeightTable, 53
- tot\_ambig\_mass
  - RoundParams, 40
  - Target, 59
- tot\_counts
  - Target, 60
- tot\_mass
  - FLD, 19
- total
  - FrequencyMatrix, 28
- total\_fpb
  - TargetTable, 64
- uniq\_counts
  - Target, 60
- unlock
  - Target, 60
- update
  - MismatchTable, 36
- update\_covar
  - TargetTable, 64
- update\_expectations
  - BiasBoss, 11
- update\_observed
  - BiasBoss, 11
- update\_target\_bias
  - Target, 60
- update\_total\_fpb
  - TargetTable, 65
- var\_sum
  - RoundParams, 41
  - Target, 60
- write\_active
  - MapParser, 33
- write\_fragment
  - BAMWriter, 8
  - SAMWriter, 44
  - Writer, 66
- Writer, 65
  - write\_fragment, 66