# eXpress

1.1.1

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

Generated by Doxygen 1.7.3

Sun Jun 24 2012 20:45:25

# Contents

# Chapter 1

# Class Index

## 1.1 Class Hierarchy

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

# Chapter 2

# Class Index

## 2.1 Class List

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

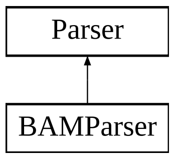
# Chapter 3

# 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**

| | |
|---|---|
| *f* | the empty Fragment 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 317 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.

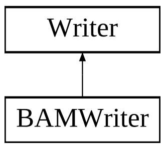
The documentation for this class was generated from the following files:

- 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**

| | |
|---|---|
| *writer* | pointer to the BAM file object |
| *sample* | specifies if a single alignment should be sampled (true) or all output with their respective probabilities (false) |

Definition at line 327 of file mapparser.cpp.

#### 3.2.2.2 BAMWriter::∼BAMWriter ( )

BAMWriter destructor flushes and deletes the Bamtools::BamWriter

Definition at line 331 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 ouptut file in BAM format along with their probabilities in the "XP" field

**Parameters**

| | |
|---|---|
| *f* | the processed Fragment to output |

Implements Writer.

Definition at line 337 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**

| | |
|---|---|
| *alpha* | 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**

| | |
|---|---|
| *outfile* | 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**

| | |
|---|---|
| *start_bias* | a vector containing the logged bias for each 5' start site in the target |
| *end_bias* | a vector containing the logged bias for each 3' end site in the target |
| *targ* | 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**

| | |
|---|---|
| *targ* | 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**

| | |
|---|---|
| *hit* | the fragment hit (alignment) |
| *mass* | the logged probabality 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**

| | |
|---|---|
| *targ* | a pointer to the initial Target object in the bundle |
| *fmt* | a pointer to the (global) FragMassTable |

Definition at line 42 of file bundles.cpp.

### 3.4.3 Member Function Documentation

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

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

**Returns**

the total number of fragments mapped to targets in the bundle

Definition at line 115 of file bundles.h.

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

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

**Parameters**

| | |
|---|---|
| *incr_amt* | the amount to increase the counts by |

Definition at line 46 of file bundles.cpp.

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

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

**Returns**

the number of targets in the bundle

Definition at line 103 of file bundles.h.

#### 3.4.3.4 std::vector<Target∗>& Bundle::targets ( ) `[inline]`

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

**Returns**

reference to the vector pointing to bundle targets

Definition at line 109 of file bundles.h.

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

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

## 3.5 BundleTable Class Reference

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

### Public Member Functions

- BundleTable ()
- ∼BundleTable ()
- const BundleSet & bundles () const
- size_t size () const
- Bundle ∗ create_bundle (Target ∗targ)
- Bundle ∗ merge (Bundle ∗b1, Bundle ∗b2)

### 3.5.1 Detailed Description

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

**Author**

Adam Roberts

**Date**

2011 Artistic License 2.0

Definition at line 129 of file bundles.h.

### 3.5.2 Constructor & Destructor Documentation

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

BundleTable constructor.

Definition at line 141 of file bundles.h.

#### 3.5.2.2 BundleTable::∼BundleTable ( )

BundleTable deconstructor. Deletes all Bundle objects.

Definition at line 51 of file bundles.cpp.

### 3.5.3 Member Function Documentation

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

a member function that returns the set of current Bundle objects

**Returns**

a reference to the unordered_set containing all current Bundle objects

Definition at line 152 of file bundles.h.

#### 3.5.3.2 Bundle ∗ BundleTable::create_bundle ( Target ∗ *targ* )

a member function that creates a new bundle, initially with only the single given Target

**Parameters**

| | |
|---|---|
| *targ* | a pointer to the only Target initially contained in the Bundle |

**Returns**

a pointer to the new Bundle object

Definition at line 59 of file bundles.cpp.

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

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

**Parameters**

| | |
|---|---|
| *b1* | a pointer to one of the Bundle objects to merge |
| *b2* | a pointer to the other Bundle object to merge |

**Returns**

a pointer to the merged Bundle object

Definition at line 66 of file bundles.cpp.

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

a member function that returns the size of the set of current Bundle objects, which is the current number of bundles

**Returns**

the current number of bundles

Definition at line 159 of file bundles.h.

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

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

## 3.6 CovarTable Class Reference

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

### Public Member Functions

- void increment (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**

| | |
|---|---|
| *targ1* | one of the targets in the pair |
| *targ2* | 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**

| | |
|---|---|
| *targ1* | one of the targets in the pair |
| *targ2* | the other target in the pair |
| *covar* | 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**

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

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**

| | |
|---|---|
| *len* | an integer for the observed FragHit length |
| *mass* | a double for the mass (logged) of the observed FragHit |

Definition at line 53 of file fld.cpp.

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

a member function that appends the FLD parameters to the end of the given file

**Parameters**

| | |
|---|---|
| *outfile* | 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**

| | |
|---|---|
| *len* | an integer for the FragHit 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
- TargID 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**

| | |
|---|---|
| *f* | the FragHit 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**

| | |
|---|---|
| *m* | a size_t specifying the number of distributions (rows) |
| *n* | a size_t specifying the number of values in each distribution (columns) |
| *alpha* | a double specifying the intial psuedo-counts (un-logged) |
| *logged* | 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**

| | |
|---|---|
| *k* | 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* | the distribution (row) |
| *j* | the value (column) |
| *incr_amt* | 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**

| | |
|---|---|
| *k* | the array position |
| *incr_amt* | 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**

| | |
|---|---|
| *k* | 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* | the distribution (row) |
| *j* | the value (column) |
| *normalized* | 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**

| | |
|---|---|
| *logged* | 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**

| | |
|---|---|
| *i* | 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 *l* ) `[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**

| | |
|---|---|
| *input_file* | string containing the path to the input SAM/BAM file |
| *output_file* | 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**

| | |
|---|---|
| *thread_-safety* | a pointer to the struct containing shared queues with the processing thread |
| *stop_at* | 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) sould be ouptut

**Parameters**

| | |
|---|---|
| *b* | 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**

| | |
|---|---|
| *alpha* | 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**

| | |
|---|---|
| *active* | 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**

| | |
|---|---|
| *file* | 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 likeihood of mismatches in the mapping given the current error model parematers

**Parameters**

| | |
|---|---|
| *f* | 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**

| | |
|---:|---|
| *f* | the fragment mapping |
| *p* | the logged posterior probablity of the alignment |
| *mass* | the logged mass of the fragment DOC |

Definition at line 166 of file mismatchmodel.cpp.

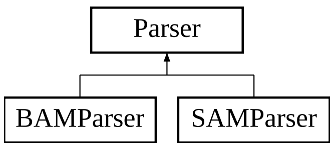
The documentation for this class was generated from the following files:

- 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 SAMParser or BAMParser. It fills Fragment objects by parsing an input file in SAM/BAM format.

**Author**

Adam Roberts

**Date**

2011 Artistic License 2.0

Definition at line 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 BAMParser, and SAMParser.

### 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**

| | |
|---|---|
| *f* | the empty Fragment to add mappings to |

**Returns**

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

Implemented in BAMParser, and SAMParser.

#### 3.17.2.3 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.

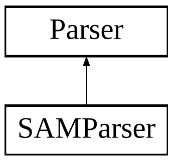
The documentation for this struct was generated from the following file:

- 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 370 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**

| | |
|---|---|
| *f* | the empty Fragment to add mappings to |

**Returns**

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

Implements Parser.

Definition at line 426 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.

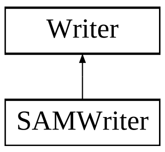
The documentation for this class was generated from the following files:

- 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)
- $\sim$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**

| | |
|---|---|
| *out* | SAM output stream |
| *sample* | specifies if a single alignment should be sampled (true) or all output with their respective probabilities (false) |

Definition at line 578 of file mapparser.cpp.

#### 3.22.2.2 SAMWriter::∼SAMWriter ( )

SAMWriter destructor flushes and deletes output stream

Definition at line 582 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 ouptut file in SAM format along with their probabilities in the "XP" field

**Parameters**

| | |
|---|---|
| *f* | the processed Fragment to output |

Implements Writer.

Definition at line 588 of file mapparser.cpp.

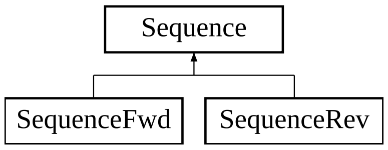
The documentation for this class was generated from the following files:

- 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**

| | |
|---|---|
| *index* | the index of the encoded character to return (assumed to be $<$ _len) |

**Returns**

the encoded character at the given index

Implemented in SequenceFwd, and SequenceRev.

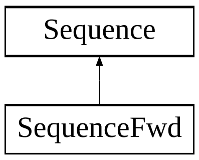
The documentation for this class was generated from the following file:

- 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 91 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**

| | |
|---|---|
| *seq* | the nucleotide sequence to encode and store |
| *rev* | 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**

| | |
|---|---|
| *other* | the Sequence object to copy |

Definition at line 29 of file sequence.cpp.

#### 3.24.2.4 SequenceFwd::∼SequenceFwd ( )

Sequence deconstructor. 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 160 of file sequence.h.

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

Sequence assignment constructor

**Parameters**

| | |
|---|---|
| *other* | the Sequence 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**

| | |
|---|---|
| *index* | 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**

| | |
|---|---|
| *seq* | the nucleotide sequence to encode and store |
| *rev* | a boolean if the sequence should be reverse complemented before encoding |

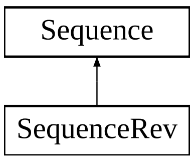
Definition at line 62 of file sequence.cpp.

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 164 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 172 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**

| | |
|---|---|
| *index* | 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 175 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**

| | |
|---|---|
| *window_size* | an unsigned integer specifying the size of the bias window surrounding fragment ends |
| *alpha* | 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**

| | |
|---|---|
| *outfile* | 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**

| | |
|---|---|
| *other* | another SeqWeightTable 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**

| | |
|---|---|
| *other* | another SeqWeightTable 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**

| | |
|---|---|
| *seq* | the target sequence the fragment hits to |
| *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**

| | |
|---|---|
| *seq* | the target sequence |
| *mass* | 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**

| | |
|---|---|
| *seq* | the target sequence (possibly reverse complemented) to which the fragment end maps |
| *i* | the index into the sequence at which to center the bias window (where the fragment starts/ends) |
| *mass* | 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**

| | |
|---|---|
| *name* | a string that stores the target name |
| *seq* | a string that stores the target sequence |
| *prob_seq* | a bool that specifies if the sequence is to be treated probablistically (for RDD detection) |
| *alpha* | a double that specifies the intial pseudo-counts (non-logged) |
| *globs* | 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**

| | |
|---|---|
| *p* | a double for the (logged) probability that the fragment was generated by this target |
| *v* | a double for the (logged) approximate variance (uncertainty) on the probability |
| *mass* | 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**

| | |
|---|---|
| *b* | a pointer to the Bundle to set this Target 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 calcualtes and returns the estimated effective length of the target (logged) using the avg bias

**Parameters**

| | |
|---|---|
| *fld* | an optional pointer to a different FLD than the global one, for thread-safety |
| *with_bias* | 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**

| | |
|---|---|
| *uniq* | a bool specifying whether or not the fragment uniquely maps to this target |
| *incr_amt* | 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**

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

**Returns**

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

Definition at line 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**

| | |
|---|---|
| *with_pseudo* | 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**

| | |
|---|---|
| *a* | 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**

| | |
|---|---|
| *bias_table* | an optional pointer to a different BiasBoss than the global one, for thread-safety |
| *fld* | an optional pointer to a different FLD 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 assignments

**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**

| | |
|---|---|
| *targ_fasta_-file* | a string storing the path to the fasta file from which to load targets |
| *targ_index* | the target-to-index map from the alignment file |
| *targ_lengths* | the target-to-length map from the alignment file |
| *prob_seqs* | a bool that specifies if the sequence is to be treated probablistically (for RDD detection) |
| *alpha* | a double that specifies the intial pseudo-counts for each bp of the targets (non-logged) |
| *alpha_map* | an optional pointer to a map object that specifies proportional weights of pseudo-counts for each target |
| *globs* | 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**

| | |
|---|---|
| *targ1* | one of the targets in the pair |
| *targ2* | 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 ( TargID *id* )

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

**Parameters**

| | |
|---|---|
| *id* | 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**

| | |
|---|---|
| *b1* | a pointer to the first Bundle to merge |
| *b2* | a pointer to the second Bundle 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**

| | |
|---|---|
| *output_dir* | the directory to output the expression file to |
| *tot_counts* | the total number of observed mapped fragments |
| *output_-varcov* | boolean specifying whether to also output the variance-covariance matrix |
| *output_-varcov* | boolean specifying whether to also output the editing p-values |

Definition at line 386 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 581 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 569 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**

| | |
|---|---|
| *targ1* | one of the targets in the pair |
| *targ2* | the other target in the pair |
| *covar* | 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**

| | |
|---|---|
| *incr_amt* | the (logged) amount to increment by |

Definition at line 575 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.

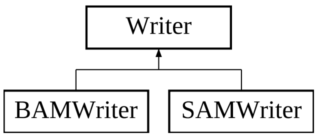
The documentation for this class was generated from the following files:

- 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 than 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 ouptut file along with their probabilities in the "XP" field

**Parameters**

| | |
|---|---|
| *f* | the processed Fragment to output |

Implemented in BAMWriter, and SAMWriter.

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

- src/mapparser.h

# Index