
# eXpress

0.9.4 BETA

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

Generated by Doxygen 1.7.3

Thu Jan 5 2012 14:34:16

# 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:

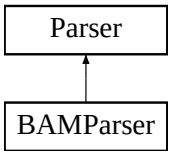
# Chapter 3

# 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 & trans_index () const
- const TransIndex & trans_lengths () const
- bool next_fragment (Fragment &f)
- void reset ()

### 3.1.1 Detailed Description

The BAMParser class fills Fragment objects by parsing an input file in BAM format.

**Author**

Adam Roberts

**Date**

2011 Artistic License 2.0

Definition at line 102 of file mapparser.h.

### 3.1.2 Constructor & Destructor Documentation

#### 3.1.2.1 BAMParser::BAMParser ( BamTools::BamReader * *reader* )

BAMParser constructor sets the reader

Definition at line 184 of file mapparser.cpp.

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

BAMParser destructor deletes the reader

Definition at line 140 of file mapparser.h.

### 3.1.3 Member Function Documentation

#### 3.1.3.1 const std::string BAMParser::header ( ) const [inline, virtual]

a member function that returns a string version of the header

**Returns**

string version of the header

Implements Parser.

Definition at line 146 of file mapparser.h.

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

a member function that loads all mappings of the next fragment

**Parameters**

| | |
|---|---|
| *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 207 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 263 of file mapparser.cpp.

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

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

**Returns**

the transcript-to-index map

Implements Parser.

Definition at line 152 of file mapparser.h.

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

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

**Returns**

the transcript-to-length map

Implements Parser.

Definition at line 158 of file mapparser.h.

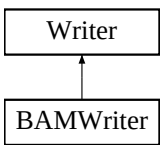
The documentation for this class was generated from the following files:

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

## 3.2 BAMWriter Class Reference

`#include <mapparser.h>`

Inheritance diagram for BAMWriter:



### Public Member Functions

- BAMWriter (BamTools::BamWriter ∗writer, bool sample)
- ∼BAMWriter ()
- void write_fragment (Fragment &f)

## 3.2.1 Detailed Description

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

**Author**

Adam Roberts

**Date**

2011 Artistic License 2.0

Definition at line 179 of file mapparser.h.

## 3.2.2 Constructor & Destructor Documentation

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

BAMWriter constructor stores a pointer to the BAM file

**Parameters**

| | |
|---|---|
| *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 273 of file mapparser.cpp.

### 3.2.2.2 BAMWriter::∼BAMWriter ( )

BAMWriter destructor flushes and deletes the Bamtools::BamWriter

Definition at line 277 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 283 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 update_expectations (const Transcript &trans)
- void normalize_expectations ()
- void update_observed (const FragHit &hit, double mass)
- double get_transcript_bias (std::vector< float > &start_bias, std::vector< float > &end_bias, const Transcript &trans) 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 220 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 paramater) |

Definition at line 228 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_transcript_bias ( std::vector< float > & *start_bias*, std::vector< float > & *end_bias*, const Transcript & *trans* ) const

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

**Parameters**

| | |
|---|---|
| *start_bias* | a vector containing the logged bias for each 5' start site in the transcript |
| *end_bias* | a vector containing the logged bias for each 3' end site in the transcript |
| *trans* | the transcript for which to calculate the logged bias |

**Returns**

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

Definition at line 303 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 255 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 332 of file biascorrection.cpp.

#### 3.3.3.5 void BiasBoss::update_expectations ( const Transcript & *trans* )

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

**Parameters**

| | |
|---|---|
| *trans* | the transcript to measure expected counts from |

Definition at line 235 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 transcript 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 263 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 (Transcript *trans)

- void incr_counts (size_t incr_amt=1)
- size_t size () const
- std::vector< Transcript * > & transcripts ()
- size_t counts () const

### 3.4.1 Detailed Description

The Bundle class keeps track of a group of transcripts that have shared ambiguous (multi-mapped) reads. Besides storing the transcript, 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 ( Transcript * *trans* )

Bundle Constructor.

**Parameters**

| | |
|---|---|
| *trans* | a pointer to the initial Transcript 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 transcripts in the bundle

**Returns**

the total number of fragments mapped to transcripts 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 transcripts in the bundle

**Returns**

the number of transcripts in the bundle

Definition at line 103 of file bundles.h.

#### 3.4.3.4 std::vector<Transcript*>& Bundle::transcripts ( ) [inline]

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

**Returns**

reference to the vector pointing to bundle transcripts

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 (Transcript *trans)
- 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 transcript 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 ( Transcript * *trans* )

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

**Parameters**

| | |
|---|---|
| *trans* | a pointer to the only Transcript 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 Transcripts 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 (TransID trans1, TransID trans2, double covar)

- double get (TransID trans1, TransID trans2)
- size_t size () const

### 3.6.1 Detailed Description

The CovarTable is a sparse matrix for storing and updating pairwise covariances between targets.

**Author**

Adam Roberts

**Date**

2011 Artistic License 2.0

Definition at line 26 of file bundles.h.

### 3.6.2 Member Function Documentation

#### 3.6.2.1 double CovarTable::get ( TransID *trans1*, TransID *trans2* )

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

**Parameters**

| | |
|---|---|
| *trans1* | one of the transcripts in the pair |
| *trans2* | the other transcript in the pair |

**Returns**

the negative of the pair's covariance (logged)

Definition at line 29 of file bundles.cpp.

#### 3.6.2.2 void CovarTable::increment ( TransID *trans1*, TransID *trans2*, double *covar* )

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

**Parameters**

| | |
|---|---|
| *trans1* | one of the transcripts in the pair |
| *trans2* | the other transcript 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 transcripts with non-zero covariance

**Returns**

the number of transcript 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
- double mean () const
- void add_val (size_t len, double mass)
- double pdf (size_t len) 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 25 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 write-locked

**Parameters**

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

Definition at line 46 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 read-locked

**Parameters**

| | |
|---|---|
| *outfile* | the file to append to |

#### 3.7.3.3 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.

#### 3.7.3.4 double FLD::mean ( ) const

a member function that returns the mean FragHit length read-locked

**Returns**

mean observed FragHit length

Definition at line 83 of file fld.cpp.

#### 3.7.3.5 double FLD::pdf ( size_t *len* ) const

a member function that returns the (logged) probability of a given FragHit length read-locked

**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 68 of file fld.cpp.

#### 3.7.3.6 string FLD::to_string ( ) const

a member function that returns a string containing the current distribution read-locked

**Returns**

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

Definition at line 90 of file fld.cpp.

#### 3.7.3.7 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 76 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

- int length () const
- PairStatus pair_status () const

### Public Attributes

- std::string name
- TransID trans_id
- Transcript * mapped_trans
- std::string seq_l
- std::string seq_r
- int left
- int right
- int mate_l
- bool left_first
- 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 37 of file fragments.h.

#### 3.8.2 Member Function Documentation

##### 3.8.2.1 int 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 95 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 106 of file fragments.h.

### 3.8.3 Member Data Documentation

#### 3.8.3.1 int FragHit::left

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

Definition at line 68 of file fragments.h.

#### 3.8.3.2 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 transcript coordinate space when true

Definition at line 88 of file fragments.h.

#### 3.8.3.3 Transcript* FragHit::mapped_trans

a public pointer to the transcript mapped to

Definition at line 52 of file fragments.h.

#### 3.8.3.4 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 81 of file fragments.h.

#### 3.8.3.5 std::string FragHit::name

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

Definition at line 42 of file fragments.h.

#### 3.8.3.6 int FragHit::right

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

Definition at line 74 of file fragments.h.

#### 3.8.3.7 std::string FragHit::seq_l

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

Definition at line 57 of file fragments.h.

#### 3.8.3.8 std::string FragHit::seq_r

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

Definition at line 62 of file fragments.h.

#### 3.8.3.9 TransID FragHit::trans_id

a public TransID for the transcript mapped to

Definition at line 47 of file fragments.h.

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

- src/fragments.h

## 3.9 Fragment Class Reference

`#include <fragments.h>`

### Public Member Functions

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

### 3.9.1 Detailed Description

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

**Author**

Adam Roberts

**Date**

2011 Artistic License 2.0

Definition at line 131 of file fragments.h.

### 3.9.2 Constructor & Destructor Documentation

#### 3.9.2.1 Fragment::~Fragment ( )

Fragment destructor deletes all FragHit objects pointed to by the Fragment

Definition at line 17 of file fragments.cpp.

### 3.9.3 Member Function Documentation

#### 3.9.3.1 bool Fragment::add_map_end ( FragHit * *f* )

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

**Parameters**

| | |
|---|---|
| *f* | the FragHit to be added |

Definition at line 30 of file fragments.cpp.

#### 3.9.3.2 const std::vector<FragHit*>& Fragment::hits ( ) const [inline]

a member function that returns FragHit multi-mappings of the fragment

**Returns**

a vector containing pointers to the FragHit multi-mappings

Definition at line 188 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 176 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 182 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 94 of file fragments.cpp.

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

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

## 3.10 FrequencyMatrix Class Reference

`#include <frequencymatrix.h>`

### Public Member Functions

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

### 3.10.1 Detailed Description

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

**Author**

Adam Roberts

**Date**

2011 Artistic License 2.0

Definition at line 24 of file frequencymatrix.h.

### 3.10.2 Constructor & Destructor Documentation

#### 3.10.2.1 FrequencyMatrix::FrequencyMatrix ( ) [inline]

dummy constructor

Definition at line 56 of file frequencymatrix.h.

#### 3.10.2.2 FrequencyMatrix::FrequencyMatrix ( size_t *m*, size_t *n*, double *alpha*, bool *logged* = true )

FrequencyMatrix constructor initializes the matrix based on the log of the given pseudo-counts

**Parameters**

| | |
|---|---|
| *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 16 of file frequencymatrix.cpp.

### 3.10.3 Member Function Documentation

#### 3.10.3.1 double FrequencyMatrix::arr ( size_t *k* ) const [inline]

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

**Parameters**

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

**Returns**

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

Definition at line 102 of file frequencymatrix.h.

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

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

**Parameters**

| | |
|---|---|
| *i* | the distribution (row) |
| *j* | the value (column) |
| *incr_amt* | the amount to increase the mass by (logged if table is logged) |

Definition at line 40 of file frequencymatrix.cpp.

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

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

**Parameters**

| | |
|---|---|
| *k* | the array position |
| *incr_amt* | the amount to increase the mass by (logged if table is logged) |

Definition at line 56 of file frequencymatrix.cpp.

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

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

**Parameters**

| | |
|---|---|
| *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 34 of file frequencymatrix.cpp.

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

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

**Parameters**

| | |
|---|---|
| *i* | the distribution (row) |
| *j* | the value (column) |

**Returns**

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

Definition at line 25 of file frequencymatrix.cpp.

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

a member function that returns the raw row sum

**Parameters**

| | |
|---|---|
| *i* | the distribution (row) |

**Returns**

a double specifying the raw row sum for the given distribution

Definition at line 109 of file frequencymatrix.h.

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

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

**Parameters**

| | |
|---|---|
| *logged* | bool specifying if the table should be converted to logged or non-logged space |

Definition at line 61 of file frequencymatrix.cpp.

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

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

## 3.11 Globals Struct Reference

`#include <main.h>`

### Public Attributes

- FLD * **fld**
- MismatchTable * **mismatch_table**
- BiasBoss * **bias_table**

### 3.11.1 Detailed Description

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

Definition at line 28 of file main.h.

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

- src/main.h

## 3.12 MismatchTable Class Reference

`#include <mismatchmodel.h>`

### Public Member Functions

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

### 3.12.1 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.

**Author**

Adam Roberts

**Date**

2011 Artistic License 2.0

Definition at line 24 of file mismatchmodel.h.

### 3.12.2 Constructor & Destructor Documentation

#### 3.12.2.1 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 18 of file mismatchmodel.cpp.

### 3.12.3 Member Function Documentation

#### 3.12.3.1 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 59 of file mismatchmodel.h.

#### 3.12.3.2 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 90 of file biascorrection.cpp.

#### 3.12.3.3 double MismatchTable::log_likelihood ( const FragHit & *f* ) const

member function returns the log likeihood of mismatches in the mapping given the current error model paramaters

**Parameters**

| | |
|---|---|
| *f* | the fragment mapping to calculate the log likelihood for |

**Returns**

the log likelihood of the mapping based on mismatches

Definition at line 25 of file mismatchmodel.cpp.

#### 3.12.3.4 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 113 of file mismatchmodel.cpp.

#### 3.12.3.5 void MismatchTable::update ( const FragHit & *f*, double *mass* )

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

**Parameters**

| | |
|---|---|
| *f* | the fragment mapping |
| *mass* | the logged mass to increase the parameters by |

Definition at line 72 of file mismatchmodel.cpp.

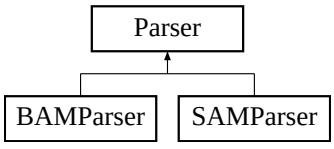
The documentation for this class was generated from the following files:

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

## 3.13 Parser Class Reference

`#include <mapparser.h>`

Inheritance diagram for Parser:




### Public Member Functions

- virtual const std::string header () const =0
- virtual const TransIndex & trans_index () const =0

- virtual const TransIndex & trans_lengths () const =0
- virtual bool next_fragment (Fragment &f)=0
- virtual void reset ()=0

### 3.13.1 Detailed Description

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

**Author**

Adam Roberts

**Date**

2011 Artistic License 2.0

Definition at line 33 of file mapparser.h.

### 3.13.2 Member Function Documentation

#### 3.13.2.1 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.

#### 3.13.2.2 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.

#### 3.13.2.3 virtual void Parser::reset ( ) [pure virtual]

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

Implemented in BAMParser.

#### 3.13.2.4 virtual const TransIndex& Parser::trans_index ( ) const [pure virtual]

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

**Returns**

the transcript-to-index map

Implemented in BAMParser, and SAMParser.

#### 3.13.2.5 virtual const TransIndex& Parser::trans_lengths ( ) const [pure virtual]

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

**Returns**

the transcript-to-length map

Implemented in BAMParser, and SAMParser.

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

- src/mapparser.h

## 3.14 ParseThreadSafety Struct Reference

`#include <mapparser.h>`

### Public Attributes

- Fragment * next_frag
- boost::mutex mut
- boost::condition_variable cond
- bool frag_clean

### 3.14.1 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 344 of file mapparser.h.

### 3.14.2 Member Data Documentation

#### 3.14.2.1 boost::condition_variable ParseThreadSafety::cond

a conditional variable where the processor waits for a new Fragment and the parser waits for the Fragment pointer to be copied by the processor

Definition at line 360 of file mapparser.h.

#### 3.14.2.2 bool ParseThreadSafety::frag_clean

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

Definition at line 366 of file mapparser.h.

#### 3.14.2.3 boost::mutex ParseThreadSafety::mut

a mutex for the conditional variable

Definition at line 354 of file mapparser.h.

#### 3.14.2.4 Fragment* ParseThreadSafety::next_frag

a pointer to the next Fragment to be processed by the main thread

Definition at line 349 of file mapparser.h.

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

- src/mapparser.h

## 3.15 PosWeightTable Class Reference

`#include <biascorrection.h>`

### Public Member Functions

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

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

### 3.15.1 Detailed Description

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

**Author**

Adam Roberts

**Date**

2011 Artistic License 2.0

Definition at line 109 of file biascorrection.h.

### 3.15.2 Constructor & Destructor Documentation

#### 3.15.2.1 PosWeightTable::PosWeightTable ( const std::vector< size_t > & *len_bins*, const std::vector< double > & *pos_bins*, double *alpha* )

PosWeightTable Constructor

**Parameters**

| | |
|---|---|
| *len_bins* | a vector of unsigned integers specifying the bin ranges for transcript lengths |
| *pos_bins* | a vector of doubles specifying the bin ranges for fractional positions |
| *alpha* | a double specifying the strength of the uniform prior (logged pseudo-counts for each paramater) |

Definition at line 128 of file biascorrection.cpp.

### 3.15.3 Member Function Documentation

#### 3.15.3.1 void PosWeightTable::append_output ( std::ofstream & *outfile* ) const

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

**Parameters**

| | |
|---|---|
| *outfile* | the file to append to |

#### 3.15.3.2 double PosWeightTable::get_weight ( size_t *len*, double *pos* ) const

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

**Parameters**

| | |
|---|---|
| *len* | the transcript length |
| *pos* | the fractional transcript position |

**Returns**

the logged bias weight for the fractional transcript position

Definition at line 166 of file biascorrection.cpp.

#### 3.15.3.3 double PosWeightTable::get_weight ( size_t *l*, size_t *p* ) const

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

**Parameters**

| | |
|---|---|
| *l* | the transcript length bin |
| *p* | the fractional transcript position bin |

**Returns**

the logged bias weight for the fractional transcript position

Definition at line 60 of file biascorrection.cpp.

#### 3.15.3.4 void PosWeightTable::increment_expected ( size_t *l*, size_t *p* )

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

**Parameters**

| | |
|---|---|
| *l* | the transcript length bin |
| *p* | the fractional transcript position bin |

Definition at line 142 of file biascorrection.cpp.

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

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

**Parameters**

| | |
|---|---|
| *len* | the transcript length |
| *pos* | the fractional transcript position |

Definition at line 135 of file biascorrection.cpp.

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

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

**Parameters**

| | |
|---|---|
| *l* | the transcript length bin |
| *p* | the fractional transcript position bin |
| *normalized_mass* | the mass (logged probabilistic assignment) of the fragment normalized by its estimated expression |

Definition at line 49 of file biascorrection.cpp.

#### 3.15.3.7 void PosWeightTable::increment_observed ( size_t *len*, double *pos*, double *normalized_mass* )

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

**Parameters**

| | |
|---|---|
| *len* | the transcript length |
| *pos* | the fractional transcript position |
| *normalized_mass* | the mass (logged probabilistic assignment) of the fragment normalized by its estimated expression |

Definition at line 153 of file biascorrection.cpp.

#### 3.15.3.8 void PosWeightTable::normalize_expected ( )

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

Definition at line 148 of file biascorrection.cpp.

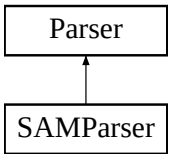
The documentation for this class was generated from the following files:

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

## 3.16 SAMParser Class Reference

`#include <mapparser.h>`

Inheritance diagram for SAMParser:




### Public Member Functions

- SAMParser (std::istream ∗in)
- const std::string header () const
- const TransIndex & trans_index () const
- const TransIndex & trans_lengths () const
- bool next_fragment (Fragment &f)

### 3.16.1 Detailed Description

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

**Author**

Adam Roberts

**Date**

2011 Artistic License 2.0

Definition at line 223 of file mapparser.h.

### 3.16.2 Constructor & Destructor Documentation

#### 3.16.2.1 SAMParser::SAMParser ( std::istream ∗ *in* )

SAMParser constructor removes the header, and parses the first line

Definition at line 303 of file mapparser.cpp.

### 3.16.3 Member Function Documentation

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

a member function that returns a string version of the header

**Returns**

string version of the header

Implements Parser.

Definition at line 271 of file mapparser.h.

#### 3.16.3.2 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 358 of file mapparser.cpp.

#### 3.16.3.3 const TransIndex& SAMParser::trans_index ( ) const [inline, virtual]

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

**Returns**

the transcript-to-index map

Implements Parser.

Definition at line 277 of file mapparser.h.

#### 3.16.3.4 const TransIndex& SAMParser::trans_lengths ( ) const [inline, virtual]

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

**Returns**

the transcript-to-length map

Implements Parser.

Definition at line 283 of file mapparser.h.

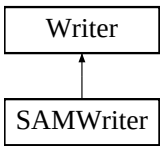
The documentation for this class was generated from the following files:

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

## 3.17 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)

### 3.17.1 Detailed Description

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

**Author**

Adam Roberts

**Date**

2011 Artistic License 2.0

Definition at line 301 of file mapparser.h.

### 3.17.2 Constructor & Destructor Documentation

#### 3.17.2.1 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 476 of file mapparser.cpp.

#### 3.17.2.2 SAMWriter::~SAMWriter ( )

SAMWriter destructor flushes and deletes output stream

Definition at line 480 of file mapparser.cpp.

### 3.17.3 Member Function Documentation

#### 3.17.3.1 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 486 of file mapparser.cpp.

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

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

## 3.18 SeqWeightTable Class Reference

`#include <biascorrection.h>`

### Public Member Functions

- SeqWeightTable (size_t window_size, double alpha)
- void increment_expected (char c)
- void normalize_expected ()
- void increment_observed (std::string &seq, double normalized_mass)
- double get_weight (const std::string &seq, int i) const
- std::string to_string () const
- void append_output (std::ofstream &outfile) const

### 3.18.1 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 32 of file biascorrection.h.

### 3.18.2 Constructor & Destructor Documentation

#### 3.18.2.1 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 paramater) |

Definition at line 29 of file biascorrection.cpp.

### 3.18.3 Member Function Documentation

#### 3.18.3.1 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 |

#### 3.18.3.2 double SeqWeightTable::get_weight ( const std::string & *seq*, int *i* ) const

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

**Parameters**

| | |
|---|---|
| *seq* | the transcript 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

#### 3.18.3.3 void SeqWeightTable::increment_expected ( char *c* )

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

**Parameters**

| | |
|---|---|
| *c* | a char representing a nucleotide that has been observed in the transcriptome |

Definition at line 34 of file biascorrection.cpp.

#### 3.18.3.4 void SeqWeightTable::increment_observed ( std::string & *seq*, double *normalized_mass* )

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

**Parameters**

| | |
|---|---|
| *seq* | a string of nucleotides in the bias window for the sequenced fragment end |
| *normalized_mass* | the mass (logged probabilistic assignment) of the fragment normalized by its estimated expression |

#### 3.18.3.5 void SeqWeightTable::normalize_expected ( )

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

Definition at line 44 of file biascorrection.cpp.

#### 3.18.3.6 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

Definition at line 73 of file biascorrection.cpp.

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

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

## 3.19 ThreadedMapParser Class Reference

`#include <mapparser.h>`

### Public Member Functions

- ThreadedMapParser (std::string input_file, std::string output_file, bool write_active)
- ~ThreadedMapParser ()
- void threaded_parse (ParseThreadSafety *thread_safety, TranscriptTable *trans_table)
- const TransIndex & trans_index ()
- const TransIndex & trans_lengths ()
- void write_active (bool b)
- void reset_reader ()

### 3.19.1 Detailed Description

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

**Author**

Adam Roberts

**Date**

2011 Artistic License 2.0

Definition at line 381 of file mapparser.h.

### 3.19.2 Constructor & Destructor Documentation

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

ThreadedMapParser constructor determines what format the input is in and initializes the correct parser.

**Parameters**

| | |
|---|---|
| *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 51 of file mapparser.cpp.

#### 3.19.2.2 ThreadedMapParser::~ThreadedMapParser ( )

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

Definition at line 115 of file mapparser.cpp.

### 3.19.3 Member Function Documentation

#### 3.19.3.1 void ThreadedMapParser::reset_reader ( ) [inline]

a member function that resets the input parser

Definition at line 439 of file mapparser.h.

#### 3.19.3.2 void ThreadedMapParser::threaded_parse ( ParseThreadSafety * *thread_safety*, TranscriptTable * *trans_table* )

a member function that drives the parse thread when all valid mappings of a fragment have been parsed, its mapped transcripts 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 locks and data with the processing thread |
| *trans_table* | a pointer to the table of Transcript objects to lookup the mapped transcripts |

Definition at line 122 of file mapparser.cpp.

#### 3.19.3.3 const TransIndex& ThreadedMapParser::trans_index ( ) [inline]

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

**Returns**

the transcript-to-index map

Definition at line 421 of file mapparser.h.

#### 3.19.3.4 const TransIndex& ThreadedMapParser::trans_lengths ( ) [inline]

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

**Returns**

the transcript-to-length map

Definition at line 427 of file mapparser.h.

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

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

**Parameters**

| | |
|---|---|
| *b* | updated write-active status |

Definition at line 434 of file mapparser.h.

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

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

## 3.20 Transcript Class Reference

`#include <transcripts.h>`

### Public Member Functions

- Transcript (const size_t id, const std::string &name, const std::string &seq, double alpha, const Globals *globs)
- const std::string & name () const
- TransID id () const
- const std::string & seq () const
- size_t length () const
- double mass () const
- double tot_mass () const
- double mass_var () const

- double est_counts () const
- double est_counts_var () 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 mass)
- void add_prob_count (double p)
- void incr_uniq_counts (size_t incr_amt=1)
- double log_likelihood (const FragHit &frag, bool with_pseudo) const
- double est_effective_length () const
- double cached_effective_length () const
- void update_transcript_bias ()

### 3.20.1 Detailed Description

The Transcript class is used to store objects for the transcripts 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 41 of file transcripts.h.

### 3.20.2 Constructor & Destructor Documentation

#### 3.20.2.1 Transcript::Transcript ( const size_t *id*, const std::string & *name*, const std::string & *seq*, double *alpha*, const Globals * *globs* )

Transcript Constructor

**Parameters**

| | |
|---|---|
| *name* | a string that stores the transcript name |
| *seq* | a string that stores the transcript sequence |
| *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 transcripts.cpp.

### 3.20.3 Member Function Documentation

#### 3.20.3.1 void Transcript::add_mass ( double *p*, 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 transcript |
| *mass* | a double specifying the (logged) mass of the fragment being mapped |

Definition at line 52 of file transcripts.cpp.

#### 3.20.3.2 void Transcript::add_prob_count ( double *p* )

a member function that increases the estimated counts and estimated count variance based on the probabilistic assignment of a fragment

**Parameters**

| | |
|---|---|
| *p* | a double for the (logged) probability that the fragment was generated by this transcript |

Definition at line 62 of file transcripts.cpp.

#### 3.20.3.3 Bundle* Transcript::bundle ( ) [inline]

a member function that returns the Bundle this Transcript is a member of

**Returns**

a pointer to the Bundle this transcript is a member of

Definition at line 229 of file transcripts.h.

#### 3.20.3.4 void Transcript::bundle ( Bundle * *b* ) [inline]

a member function that set the Bundle this Transcript is a member of

**Parameters**

| | |
|---|---|
| *b* | a pointer to the Bundle to set this Transcript as a member of |

Definition at line 235 of file transcripts.h.

#### 3.20.3.5 double Transcript::cached_effective_length ( ) 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 transcript calculated

Definition at line 122 of file transcripts.cpp.

#### 3.20.3.6 double Transcript::est_counts ( ) const [inline]

a member function that returns the current (logged) estimated counts not valid in the first online EM round

**Returns**

estimated counts

Definition at line 199 of file transcripts.h.

#### 3.20.3.7 double Transcript::est_counts_var ( ) const [inline]

a member function that returns the current (logged) variance on estimated counts not valid in the first online EM round

**Returns**

variance estimated counts

Definition at line 206 of file transcripts.h.

#### 3.20.3.8 double Transcript::est_effective_length ( ) const

a member function that calcualtes and returns the effective length of the transcript (logged)

**Returns**

the effective length of the transcript calculated as $\tilde{l} = \sum_{l=1}^{L(t)} \sum_{i=1}^{L(t)} D(l) b_5[i] * b_3[i+l]$
a member function that calcualtes and returns the estimated effective length of the transcript (logged) using the avg bias
the estimated effective length of the transcript calculated as $\tilde{l} = \bar{bias} \sum_{l=1}^{L(t)} D(l)(L(t) - l + 1)$

Definition at line 108 of file transcripts.cpp.

#### 3.20.3.9 TransID Transcript::id ( ) const [inline]

a member function that returns the transcript id

**Returns**

TransID transcript ID

Definition at line 162 of file transcripts.h.

#### 3.20.3.10 void Transcript::incr_uniq_counts ( size_t *incr_amt* = 1 ) [inline]

a member function that increases the ccount of fragments uniquely mapped to this transcript

**Parameters**

| | |
|---|---|
| *incr_amt* | a size_t to increase the counts by |

Definition at line 255 of file transcripts.h.

#### 3.20.3.11 size_t Transcript::length ( ) const [inline]

a member function that returns the transcript length

**Returns**

transcript length

Definition at line 173 of file transcripts.h.

#### 3.20.3.12 double Transcript::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 transcript

**Parameters**

| | |
|---|---|
| *frag* | a FragHit to return the likelihood of being originated from this transcript |
| *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 transcript

Definition at line 77 of file transcripts.cpp.

#### 3.20.3.13 double Transcript::mass ( ) const [inline]

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

**Returns**

logged mass

Definition at line 179 of file transcripts.h.

#### 3.20.3.14 double Transcript::mass_var ( ) const [inline]

a member function that returns the current (logged) variance

**Returns**

logged mass variance

Definition at line 192 of file transcripts.h.

#### 3.20.3.15 const std::string& Transcript::name ( ) const [inline]

a member function that returns the transcript name

**Returns**

string containing transcript name

Definition at line 156 of file transcripts.h.

#### 3.20.3.16 void Transcript::round_reset ( )

a member function that readies the transcript object for the next round of batch EM
Definition at line 69 of file transcripts.cpp.

#### 3.20.3.17 const std::string& Transcript::seq ( ) const [inline]

a member function that returns the transcript sequence

**Returns**

string containing transcript sequence

Definition at line 167 of file transcripts.h.

#### 3.20.3.18 size_t Transcript::tot_counts ( ) const [inline]

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

**Returns**

total fragment count

Definition at line 217 of file transcripts.h.

#### 3.20.3.19 double Transcript::tot_mass ( ) const [inline]

a member function that returns the total (logged) mass of all fragments mapped to the transcript

**Returns**

logged total mass

Definition at line 185 of file transcripts.h.

#### 3.20.3.20 size_t Transcript::uniq_counts ( ) const [inline]

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

**Returns**

unique fragment count

Definition at line 223 of file transcripts.h.

#### 3.20.3.21 void Transcript::update_transcript_bias ( )

a member function that causes the transcript bias to be re-calculated by the _bias_table based on curent parameters

Definition at line 146 of file transcripts.cpp.

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

- src/transcripts.h
- src/transcripts.cpp

## 3.21 TranscriptTable Class Reference

```
#include <transcripts.h>
```

### Public Member Functions

- TranscriptTable (const std::string &trans_fasta_file, const TransIndex &trans_-index, const TransIndex &trans_lengths, double alpha, const Globals ∗globs)
- ∼TranscriptTable ()
- Transcript ∗ get_trans (TransID id)
- void round_reset ()
- size_t size () const
- void update_covar (TransID trans1, TransID trans2, double covar)
- double get_covar (TransID trans1, TransID trans2)
- size_t covar_size () const
- Bundle ∗ merge_bundles (Bundle ∗b1, Bundle ∗b2)
- size_t num_bundles ()
- void threaded_bias_update ()
- void output_results (std::string output_dir, size_t tot_counts, bool output_varcov)

### 3.21.1 Detailed Description

The TranscriptTable class is used to keep track of the Transcript objects for a run. The constructor parses a fasta file to generate the Transcript 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 306 of file transcripts.h.

### 3.21.2 Constructor & Destructor Documentation

#### 3.21.2.1 TranscriptTable::TranscriptTable ( const std::string & *trans_fasta_file,* const TransIndex & *trans_index,* const TransIndex & *trans_lengths,* double *alpha,* const Globals ∗ *globs* )

TranscriptTable Constructor

**Parameters**

| | |
|---|---|
| *trans_-fasta_file* | a string storing the path to the fasta file from which to load transcripts |
| *trans_index* | the transcript-to-index map from the alignment file |
| *trans_-lengths* | the transcript-to-length map from the alignment file |
| *alpha* | a double that specifies the intial pseudo-counts for each bp of the transcripts (non-logged) |
| *globs* | a pointer to the struct containing pointers to the global parameter tables (bias_table, mismatch_table, fld) |

Definition at line 156 of file transcripts.cpp.

#### 3.21.2.2 TranscriptTable::~TranscriptTable ( )

TranscriptTable Destructor deletes all of the transcript objects in the table

Definition at line 230 of file transcripts.cpp.

### 3.21.3 Member Function Documentation

#### 3.21.3.1 size_t TranscriptTable::covar_size ( ) const [inline]

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

**Returns**

the number of transcript pairs with non-zero covariance

Definition at line 400 of file transcripts.h.

#### 3.21.3.2 double TranscriptTable::get_covar ( TransID *trans1*, TransID *trans2* ) [inline]

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

**Parameters**

| | |
|---|---|
| *trans1* | one of the transcripts in the pair |
| *trans2* | the other transcript in the pair |

**Returns**

the negative of the pair's covariance (logged)

Definition at line 394 of file transcripts.h.

#### 3.21.3.3 Transcript * TranscriptTable::get_trans ( TransID *id* )

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

**Parameters**

| | |
|---|---|
| *id* | of the transcript queried |

**Returns**

pointer to the transcript wit the given id

Definition at line 260 of file transcripts.cpp.

#### 3.21.3.4 Bundle * TranscriptTable::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 265 of file transcripts.cpp.

#### 3.21.3.5 size_t TranscriptTable::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 274 of file transcripts.cpp.

#### 3.21.3.6 void TranscriptTable::output_results ( std::string *output_dir*, size_t *tot_counts*, bool *output_varcov* )

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

**Parameters**

| | |
|---|---|
| *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 |

Definition at line 357 of file transcripts.cpp.

#### 3.21.3.7 void TranscriptTable::round_reset ( )

a member function that readies all Transcript objects in the table for the next round of batch EM

Definition at line 279 of file transcripts.cpp.

#### 3.21.3.8 size_t TranscriptTable::size ( ) const [inline]

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

**Returns**

number of transcripts in the table

Definition at line 376 of file transcripts.h.

#### 3.21.3.9 void TranscriptTable::threaded_bias_update ( )

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

Definition at line 287 of file transcripts.cpp.

#### 3.21.3.10 void TranscriptTable::update_covar ( TransID *trans1*, TransID *trans2*, double *covar* ) [inline]

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

**Parameters**

| | |
|---|---|
| *trans1* | one of the transcripts in the pair |
| *trans2* | the other transcript in the pair |
| *covar* | a double specifying the amount to increase the pair's covariance by (logged) |

Definition at line 385 of file transcripts.h.

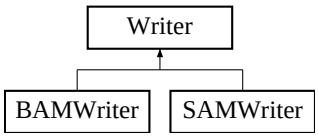
The documentation for this class was generated from the following files:

- src/transcripts.h
- src/transcripts.cpp

## 3.22 Writer Class Reference

`#include <mapparser.h>`

Inheritance diagram for Writer:



### Public Member Functions

- virtual void write_fragment (Fragment &f)=0

## 3.22.1 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 76 of file mapparser.h.

## 3.22.2 Member Function Documentation

### 3.22.2.1 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