eXpress

"1.5"

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

Generated by Doxygen 1.7.3

Sun Dec 8 2013 11:09:13

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

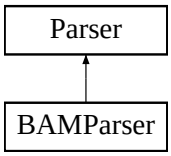
Class Documentation

3.1 BAMParser Class Reference

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

`#include <mapparser.h>`

Inheritance diagram for BAMParser:



Public Member Functions

- BAMParser (BamTools::BamReader *reader)

  BAMParser constructor sets the reader.

- const std::string header () const

  An accessor for the header string.

- bool next_fragment (Fragment &f)

  A member function that loads all mappings of the next fragment into the given Fragment object.

- void reset ()

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

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 127 of file mapparser.h.

3.1.2 Constructor & Destructor Documentation

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

BAMParser constructor sets the reader.

Parameters

reader	a pointer to the BamReader object that will directly parse the BAM file.

Definition at line 239 of file mapparser.cpp.

3.1.3 Member Function Documentation

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

An accessor for the header string.

Returns

The header string.

Implements Parser.

Definition at line 152 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 into the given Fragment object.

Parameters

f	the empty Fragment to add mappings to.

Returns

True iff more reads remain in the BAM file/stream.

Implements Parser.

Definition at line 261 of file mapparser.cpp.

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

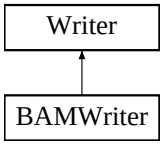
- src/mapparser.h
- src/mapparser.cpp

3.2 BAMWriter Class Reference

The BAMWriter class writes Fragment objects back to file in BAM format with per-mapping probabilistic assignments, or by sampling a single mapping based on assignment probabilities.

`#include <mapparser.h>`

Inheritance diagram for BAMWriter:



Public Member Functions

- BAMWriter (BamTools::BamWriter *writer, bool sample)

  BAMWriter constructor stores a pointer to the BamTools::BamWriter object that will directly write to the BAM file.

- ~BAMWriter ()

  BAMWriter destructor closes the BamTools::BamWriter object.

- void write_fragment (Fragment &f)

  A member function that writes the mappings to the output BAM file.

3.2.1 Detailed Description

The BAMWriter class writes Fragment objects back to file in BAM format with per-mapping probabilistic assignments, or by sampling a single mapping based on assignment probabilities.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 226 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 BamTools::BamWriter object that will directly write to the BAM file.

Parameters

writer	pointer to the BamTools::BamWriter objected assocaited with the output BAM file.
sample	specifies if a single alignment should be sampled based on posteriors (true) or all output with their respective posterior

probabilities (false).

Definition at line 550 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 the mappings to the output BAM file.

If _sample is true, a only one alignment is output, otherwise all mappings are output along with their probabilities in the "XP" field.

Parameters

f	the processed Fragment to output alignments of.

Implements Writer.

Definition at line 559 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

The BiasBoss class keeps track of sequence-specific and positional bias.

`#include <biascorrection.h>`

Public Member Functions

- BiasBoss (size_t order, double alpha)

  BiasBoss Constructor.

- BiasBoss (size_t order, std::string param_file_name)

  A second constructor that loads the distributions from a parameter file.

- size_t order () const

  An accessor for the order of the Markov chains used to model the sequences.

- void copy_observations (const BiasBoss &other)

  A member function that copies the observed parameters from another BiasBoss.

- void copy_expectations (const BiasBoss &other)

  A member function that copies the expected parameters from another BiasBoss.

- void 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 assuming uniform abundance of and coverage accross the target's sequence.

- void normalize_expectations ()

  A member function that normalizes the expected counts and fills in the lower-ordered marginals.

- void update_observed (const FragHit &hit, double mass)

  A member function that updates the observed parameters given a fragment mapping to a target and its logged probabilistic assignment value.

- double 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.

- void append_output (std::ofstream &outfile) const

  A member function that appends the 5' and 3' bias parameters to the given file, formatted in tables for easy readability.

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 138 of file biascorrection.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 BiasBoss::BiasBoss (size_t *order*, double *alpha*)

BiasBoss Constructor.

Parameters

order	a size_t specifying the order of the Markov chains used to model the sequences.
alpha	a double specifying the strength of the uniform prior (logged pseudo-counts for each parameter).

Definition at line 185 of file biascorrection.cpp.

3.3.2.2 BiasBoss::BiasBoss (size_t *order*, std::string *param_file_name*)

A second constructor that loads the distributions from a parameter file.

Note that the values should not be modified after using this constructor.

Parameters

order	a size_t specifying the order to use for the Markov chains modelling the sequence. Must match file.
param_file_name	a string specifying the path to the parameter file.

3.3.3 Member Function Documentation

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

A member function that appends the 5' and 3' bias parameters to the given file, formatted in tables for easy readability.

Parameters

outfile	the file to append to.

3.3.3.2 void BiasBoss::copy_expectations (const BiasBoss & *other*)

A member function that copies the expected parameters from another BiasBoss.

Parameters

other	a BiasBoss to copy the parameters from.

Definition at line 202 of file biascorrection.cpp.

3.3.3.3 void BiasBoss::copy_observations (const BiasBoss & *other*)

A member function that copies the observed parameters from another BiasBoss.

Parameters

other	a BiasBoss to copy the parameters from.

Definition at line 197 of file biascorrection.cpp.

3.3.3.4 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 bias.

Returns

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

Definition at line 244 of file biascorrection.cpp.

3.3.3.5 size_t BiasBoss::order ( ) const [inline]

An accessor for the order of the Markov chains used to model the sequences.

Returns

The order of the Markov chains used to model the sequences.

Definition at line 177 of file biascorrection.h.

3.3.3.6 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 assuming uniform abundance of and coverage accross the target's sequence.

Parameters

targ	the target to measure expected counts from

Definition at line 207 of file biascorrection.cpp.

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

A member function that updates the observed parameters given a fragment mapping to a target and its logged probabilistic assignment value.

Parameters

hit	the fragment hit (alignment).
mass	the logged probabality of the mapping, which is the amount to increment the observed counts by.

Definition at line 228 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

The Bundle class keeps track of a group of targets that have shared ambiguous (multi-mapped) reads.

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

Public Member Functions

- Bundle (Target *targ)

  Bundle Constructor.

- const Bundle * get_rep () const

  A private method for returning the root of the merge tree that this bundle is a node in.

- void incr_counts (size_t incr_amt=1)

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

- void incr_mass (double incr_amt)

  A member function that increases the total bundle mass (logged) by a given amount.

- void reset_mass ()

  A member function that resets the Bundle mass to (log) 0.

- size_t size () const

  An accessor for the number of Targets in the bundle.

- const std::vector< Target * > * targets () const

  An accessor for a pointer to the vector of pointers to Targets in the bundle.

- size_t counts () const

  An accessor for the the total number of observed fragments mapped to targets in the bundle.

- double mass () const

  An accessor for the the total mass of observed fragments mapped to targets in the bundle (logged), including the initial pseudo-mass.

Friends

- class BundleTable

3.4.1 Detailed Description

The Bundle class keeps track of a group of targets that have shared ambiguous (multi-mapped) reads.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 76 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.

Definition at line 34 of file bundles.cpp.

3.4.3 Member Function Documentation

3.4.3.1 size_t Bundle::counts ( ) const

An accessor for the 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 72 of file bundles.cpp.

3.4.3.2 const Bundle* Bundle::get_rep ( ) const

A private method for returning the root of the merge tree that this bundle is a node in.

Returns

A pointer to the bundle at the root of the merge tree for this bundle.

3.4.3.3 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 49 of file bundles.cpp.

3.4.3.4 void Bundle::incr_mass (double *incr_amt*)

A member function that increases the total bundle mass (logged) by a given amount.

Parameters

incr_amt	the amount to increase the mass by (logged).

Definition at line 58 of file bundles.cpp.

3.4.3.5 double Bundle::mass ( ) const

An accessor for the the total mass of observed fragments mapped to targets in the bundle (logged), including the initial pseudo-mass.

Returns

The total mass of fragments mapped to targets in the bundle.

Definition at line 80 of file bundles.cpp.

3.4.3.6 void Bundle::reset_mass ( )

A member function that resets the Bundle mass to (log) 0.

Call is not passed on to _merged_into.

Definition at line 67 of file bundles.cpp.

3.4.3.7 size_t Bundle::size ( ) const

An accessor for the number of Targets in the bundle.

Returns

The number of Targets in the bundle.

Definition at line 41 of file bundles.cpp.

3.4.3.8 const std::vector<Target*>* Bundle::targets ( ) const [inline]

An accessor for a pointer to the vector of pointers to Targets in the bundle.

The returned value does not outlive this.

Returns

Pointer to the vector pointing to bundle Targets.

Definition at line 143 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

The BundleTable class keeps track of the Bundle objects for a given run.

`#include <bundles.h>`

Public Member Functions

- BundleTable ()

  BundleTable Constructor.

- ~BundleTable ()

  BundleTable deconstructor.

- const BundleSet & bundles () const

  A member function that returns the set of current Bundle objects.

- size_t size () const

  An accessor for the current number of Bundles.

- Bundle * create_bundle (Target *targ)

  A member function that creates a new Bundle, initially containing only the single given Target.

- Bundle * merge (Bundle *b1, Bundle *b2)

  A member function that merges two Bundle objects into one.

- void collapse ()

  Collapses the merge tree so that all targets are placed in the target list of the root node and all other nodes are deleted.

- bool threadsafe_mode () const

  Accessor for whether or not the BundleTable is in threadsafe mode.

- void threadsafe_mode (bool mode)

  Mutator for threadsafe mode.

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.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 168 of file bundles.h.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 BundleTable::~BundleTable ( )

BundleTable deconstructor.

Deletes all Bundle objects.

Definition at line 91 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.

The returned object does not outlive this.

Returns

A reference to the unordered_set containing all current Bundle objects.

Definition at line 203 of file bundles.h.

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

A member function that creates a new Bundle, initially containing 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 104 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 moved 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 110 of file bundles.cpp.

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

An accessor for the current number of Bundles.

Returns

The current number of Bundles.

Definition at line 208 of file bundles.h.

3.5.3.5 bool BundleTable::threadsafe_mode ( ) const [inline]

Accessor for whether or not the BundleTable is in threadsafe mode.

Returns

True if the BundleTable is in threadsafe mode.

Definition at line 233 of file bundles.h.

3.5.3.6 void BundleTable::threadsafe_mode (bool *mode*) [inline]

Mutator for threadsafe mode.

Parameters

mode	bool specifying if threadsafe mode should be enabled (true) or disabled (false)

Definition at line 239 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

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

`#include <bundles.h>`

Public Member Functions

- CovarTable ()

  CovarTable Constructor.

- void increment (TargID targ1, TargID targ2, double covar)

  A member function that increases the covariance between two targets by the specified amount (logged).

- double get (TargID targ1, TargID targ2)

  A member function that returns the covariance between two targets.

- size_t size () const

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

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 28 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.

The 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 25 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 the true covariance is 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, positive).

Definition at line 16 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 64 of file bundles.h.

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

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

3.7 DirectionDetector Class Reference

The DirectionDetector class keeps track of the observed fragment directions (forward-reverse or reverse-forward) and whether they are paired or single-end.

`#include <directiondetector.h>`

Public Member Functions

- DirectionDetector ()

  The DirectionDetector constructor sets all counts to 0.

- void add_fragment (Fragment *f)

  Adds counts for the alignments of the given Fragment.

- bool report_if_improper_direction ()

  Throws a warning if a disproportionate number of alignments are in one direction and the proper flag has not been specified.

3.7.1 Detailed Description

The DirectionDetector class keeps track of the observed fragment directions (forward-reverse or reverse-forward) and whether they are paired or single-end. It can then determine if the numbers in each direction are distributed disproportionately and throw a warning if the proper direction flag has not been specified. Unable to detect when a directional flag has been chosen incorrectly since the incompatible alignments will have been discarded.

Author

Adam Roberts

Date

2012 Artistic License 2.0

Definition at line 29 of file directiondetector.h.

3.7.2 Member Function Documentation

3.7.2.1 void DirectionDetector::add_fragment (Fragment * f)

Adds counts for the alignments of the given Fragment.

Parameters

f	a pointer to the Fragment to count the direction of its alignments.

Definition at line 20 of file directiondetector.cpp.

3.7.2.2 bool DirectionDetector::report_if_improper_direction ( )

Throws a warning if a disproportionate number of alignments are in one direction and the proper flag has not been specified.

Returns

True iff a warning is thrown.

Definition at line 44 of file directiondetector.cpp.

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

- src/directiondetector.h
- src/directiondetector.cpp

3.8 FragHit Class Reference

The FragHit struct stores the information for a single fragment alignment.

`#include <fragments.h>`

Public Member Functions

- FragHit (ReadHit *h)
  FragHit constructor for single-end read.
- FragHit (ReadHit *l, ReadHit *r)
  Fraghit constructor for paired-end read.
- std::string frag_name () const
  Accessor for the name of the fragment.
- HitParams * params ()
  Accessor for the hit parameters (likelihood, etc.).
- const HitParams * params () const
  Const accessor for the hit parameters (likelihood, etc.).
- Target * target () const
  Accessor for a pointer to the Target object the fragment is aligned to.
- void target (Target *target)
  Mutator for a pointer to the Target object the fragment is aligned to.

- const std::vector< const Target * > * neighbors () const

  Accessor for a pointer to the vector of neighbors to the target.

- void neighbors (const std::vector< const Target * > &neighbors)

  Mutator for a vector of neighbors to the target.

- TargID target_id () const

  Accessor for the ID of the target the fragment is aligned to.

- size_t left () const

  Accessor for the leftmost position aligned to (0-based).

- size_t right () const

  Accessor for one position past the rightmost position aligned to (0-based).

- size_t length () const

  Accessor for the length of the fragment alignment.

- const ReadHit * left_read () const

  Const accessor for the alignment of the read at the leftmost (5') end of the fragment in target coordinates or NULL if it was not sequenced.

- const ReadHit * right_read () const

  Const accessor for the alignment of the read at the rightmost (3') end of the fragment in target coordinates or NULL if it was not sequenced.

- const ReadHit * first_read () const

  Const accessor for the alignment of the first (or only) read sequenced in the fragment.

- const ReadHit * second_read () const

  Const accessor for the alignment of the second read sequenced in the fragment.

- ReadHit * left_read ()

  Accessor for the alignment of the read at the leftmost (5') end of the fragment in target coordinates or NULL if it was not sequenced.

- ReadHit * right_read ()

  Accessor for the alignment of the read at the rightmost (3') end of the fragment in target coordinates or NULL if it was not sequenced.

- ReadHit * first_read ()

  Accessor for the alignment of the first (or only) read sequenced in the fragment.

- ReadHit * second_read ()

  Accessor for the alignment of the second read sequenced in the fragment.

- PairStatus pair_status () const

  A member function returning whether the mapping is PAIRED, LEFT_ONLY, or RIGHT_-ONLY, as defined in the PairStatus enum definition.

3.8.1 Detailed Description

The FragHit struct stores the information for a single fragment alignment.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 141 of file fragments.h.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 FragHit::FragHit (ReadHit * h) [inline]

FragHit constructor for single-end read.

Parameters

h	pointer to the ReadHit struct for the single-end read.

Definition at line 168 of file fragments.h.

3.8.2.2 FragHit::FragHit (ReadHit * l, ReadHit * r) [inline]

Fraghit constructor for paired-end read.

Parameters

l	pointer to the ReadHit struct for the upstream (left) read.
r	pointer to the ReadHit struct for the downstream (right) read.

Definition at line 180 of file fragments.h.

3.8.3 Member Function Documentation

3.8.3.1 const ReadHit* FragHit::first_read ( ) const [inline]

Const accessor for the alignment of the first (or only) read sequenced in the fragment.

Returns

A const pointer to the alignment of the first read.

Definition at line 307 of file fragments.h.

3.8.3.2 ReadHit* FragHit::first_read ( ) [inline]

Accessor for the alignment of the first (or only) read sequenced in the fragment.

Returns

A pointer to the alignment of the first read.

Definition at line 349 of file fragments.h.

3.8.3.3 std::string FragHit::frag_name ( ) const [inline]

Accessor for the name of the fragment.

Returns

The name of the fragment.

Definition at line 192 of file fragments.h.

3.8.3.4 size_t FragHit::left ( ) const [inline]

Accessor for the leftmost position aligned to (0-based).

Returns

The leftmost position aligned to in the target.

Definition at line 251 of file fragments.h.

3.8.3.5 ReadHit* FragHit::left_read ( ) [inline]

Accessor for the alignment of the read at the leftmost (5') end of the fragment in target coordinates or NULL if it was not sequenced.

Returns

A pointer to the 5' read alignment.

Definition at line 333 of file fragments.h.

3.8.3.6 const ReadHit* FragHit::left_read ( ) const [inline]

Const accessor for the alignment of the read at the leftmost (5') end of the fragment in target coordinates or NULL if it was not sequenced.

Returns

A const pointer to the 5' read alignment.

Definition at line 285 of file fragments.h.

3.8.3.7 size_t FragHit::length ( ) const [inline]

Accessor for the length of the fragment alignment.

Returns 0 if the fragment is single-end.

Returns

Length of fragment mapping.

Definition at line 274 of file fragments.h.

3.8.3.8 const std::vector<const Target*>* FragHit::neighbors ( ) const [inline]

Accessor for a pointer to the vector of neighbors to the target.

Experimental.

Returns

A pointer to the vector of neighbors.

Definition at line 228 of file fragments.h.

3.8.3.9 void FragHit::neighbors (const std::vector< const Target * > & *neighbors*) [inline]

Mutator for a vector of neighbors to the target.

Experimental.

Parameters

neighbors	a vector of neighbors to the target.

Definition at line 233 of file fragments.h.

3.8.3.10 PairStatus FragHit::pair_status ( ) const [inline]

A member function returning whether the mapping is PAIRED, LEFT_ONLY, or RIGHT_-ONLY, as defined in the PairStatus enum definition.

Returns

The pair status of the mapping.

Definition at line 365 of file fragments.h.

3.8.3.11 HitParams* FragHit::params ( ) [inline]

Accessor for the hit parameters (likelihood, etc.).

Returns

A pointer to the hit parameters.

Definition at line 203 of file fragments.h.

3.8.3.12 const HitParams* FragHit::params ( ) const [inline]

Const accessor for the hit parameters (likelihood, etc.).

Returns

A const pointer to the hit parameters.

Definition at line 208 of file fragments.h.

3.8.3.13 size_t FragHit::right ( ) const [inline]

Accessor for one position past the rightmost position aligned to (0-based).

Returns

One past the rightmost position aligned to in the target.

Definition at line 262 of file fragments.h.

3.8.3.14 ReadHit* FragHit::right_read ( ) [inline]

Accessor for the alignment of the read at the rightmost (3') end of the fragment in target coordinates or NULL if it was not sequenced.

Returns

A pointer to the 3' read alignment.

Definition at line 341 of file fragments.h.

3.8.3.15 const ReadHit* FragHit::right_read ( ) const [inline]

Const accessor for the alignment of the read at the rightmost (3') end of the fragment in target coordinates or NULL if it was not sequenced.

Returns

A const pointer to the 3' read alignment.

Definition at line 296 of file fragments.h.

3.8.3.16 ReadHit* FragHit::second_read ( ) [inline]

Accessor for the alignment of the second read sequenced in the fragment.

Returns NULL if single-end.

Returns

A pointer to the alignment of the second read.

Definition at line 357 of file fragments.h.

3.8.3.17 const ReadHit* FragHit::second_read ( ) const [inline]

Const accessor for the alignment of the second read sequenced in the fragment.

Returns NULL if single-end.

Returns

A const pointer to the alignment of the second read.

Definition at line 319 of file fragments.h.

3.8.3.18 Target* FragHit::target ( ) const [inline]

Accessor for a pointer to the Target object the fragment is aligned to.

Returns

A pointer to the Target aligned to.

Definition at line 213 of file fragments.h.

3.8.3.19 void FragHit::target (Target * *target*) [inline]

Mutator for a pointer to the Target object the fragment is aligned to.

Parameters

target	a pointer to the Target aligned to.

Definition at line 220 of file fragments.h.

3.8.3.20 TargID FragHit::target_id ( ) const [inline]

Accessor for the ID of the target the fragment is aligned to.

Returns

The ID of the target aligned to.

Definition at line 240 of file fragments.h.

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

- src/fragments.h

3.9 Fragment Class Reference

The Fragment class stores information for all alignments of a single fragment.

`#include <fragments.h>`

Public Member Functions

- Fragment (Library *lib)

  Fragment Constructor.

- ~Fragment ()

  Fragment destructor deletes all FragHit and ReadHit objects pointed to by the Fragment.

- const Library * lib ()

  Accessor for the global variables associated with the library this fragment is from.

- bool add_map_end (ReadHit *r)

  A member function that adds a new ReadHit to the Fragment.

- const std::string & name () const

  A member function that returns a reference to the "Query Template Name".

- size_t num_hits () const

  An accessor for the number of valid alignments of the fragment.

- FragHit * operator[ ] (size_t i) const

  An accessor for a pointer to the FragHit at the given index.

- const std::vector< FragHit * > & hits () const

  Accessor for the FragHit objects associated with the fragment.

- const FragHit * sample_hit () const

  A member function that returns a single FragHit of the fragment sampled at random based on the probabalistic assignments.

- void mass (double m)

  Mutator for the mass of the fragment according to the forgetting factor.

- double mass () const

  An accessor for the mass of the fragment according to the forgetting factor.

- void sort_hits ()

  A member function that sorts the FragHits by the TargID of the targets they are aligned to.

- bool paired () const

  An accessor that returns true iff the Fragment has paired alignments.

3.9.1 Detailed Description

The Fragment class stores information for all alignments 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 384 of file fragments.h.

3.9.2 Member Function Documentation

3.9.2.1 bool Fragment::add_map_end (ReadHit * *r*)

A member function that adds a new ReadHit to the Fragment.

If it is the first ReadHit, it sets the Fragment name. If the fragment is not paired, a FragHit is created and added to _frag_hits. Otherwise, add_open_mate is called.

Parameters

r	a pointer to the ReadHit to be added.

Returns

True iff the read name matches the Fragment name or it is the first read.

Definition at line 28 of file fragments.cpp.

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

Accessor for the FragHit objects associated with the fragment.

Returned value does not outlive this.

Returns

Reference to a vector containing pointers to the FragHits.

Definition at line 464 of file fragments.h.

3.9.2.3 const Library* Fragment::lib ( ) [inline]

Accessor for the global variables associated with the library this fragment is from.

Pointer outlives this.

Definition at line 432 of file fragments.h.

3.9.2.4 void Fragment::mass (double *m*) [inline]

Mutator for the mass of the fragment according to the forgetting factor.

Parameters

m	a double representing the value to set to the mass to.

Definition at line 476 of file fragments.h.

3.9.2.5 double Fragment::mass ( ) const [inline]

An accessor for the mass of the fragment according to the forgetting factor.

Returns

The mass of the fragment.

Definition at line 482 of file fragments.h.

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

A member function that returns a reference to the "Query Template Name".

Returns

Reference to the SAM "Query Template Name" (fragment name).

Definition at line 447 of file fragments.h.

3.9.2.7 size_t Fragment::num_hits ( ) const [inline]

An accessor for the number of valid alignments of the fragment.

Returns

Number of valid alignments for fragment.

Definition at line 452 of file fragments.h.

3.9.2.8 FragHit* Fragment::operator[] (size_t *i*) const [inline]

An accessor for a pointer to the FragHit at the given index.

Parameters

i	index of the FragHit requested.

Returns

A pointer to the FragHit at the given index.

Definition at line 458 of file fragments.h.

3.9.2.9 bool Fragment::paired ( ) const [inline]

An accessor that returns true iff the Fragment has paired alignments.

Returns

True iff the Fragment has paired alignments.

Definition at line 492 of file fragments.h.

3.9.2.10 const FragHit * Fragment::sample_hit ( ) const

A member function that returns a single FragHit of the fragment sampled at random based on the probabalistic assignments.

Returned value does not outlive this.

Returns

A randomly sampled FragHit.

Definition at line 75 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

frequencymatrix.h express

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

Public Member Functions

- FrequencyMatrix ()

  Dummy FrequencyMatrix Constructor.

- FrequencyMatrix (size_t m, size_t n, T alpha, bool logged=true)

  FrequencyMatrix constructor initializes the matrix values to the given pseudo-counts.

- T operator() (size_t i, size_t j, bool normalized=true) const

  An accessor for the frequency at a given position in the matrix (logged if table is logged).

- T operator() (size_t k, bool normalized=true) const

  An accessor for the frequency at a given position in the flattened matrix (logged if table is logged).

- void increment (size_t i, size_t j, T incr_amt)

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

- void 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).

- T sum (size_t i) const

  An accessor for the row sum (normalizer), (logged if table is logged).

- size_t argmax (size_t i) const

  A member function that finds and returns the argmax (index of mode) of the given distribution.

- void set_logged (bool logged)

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

- void fix ()

  A member function that normalizes and "locks" the matrix values so that no changes can be made.

- bool is_fixed () const

  An accessor for the value of _fixed, which specifies whether or not the matrix has been fixed (irrevocable).

3.10.1 Detailed Description

template<class T> class FrequencyMatrix< T >

frequencymatrix.h express Created by Adam Roberts on 4/23/11. Copyright 2011 Adam Roberts. All rights reserved. The FrequencyMatrix class keeps track of the frequency parameters in order to allow for constant-time probability look-ups and updates. The table is 2D to allow multiple distributions to be stored in one FrequencyMatrix. The first dimension (rows) are the different distributions. Values are stored in log space by default.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 27 of file frequencymatrix.h.

3.10.2 Constructor & Destructor Documentation

3.10.2.1 template<class T> FrequencyMatrix< T >::FrequencyMatrix (size_t *m*, size_t *n*, T *alpha*, bool *logged* = `true`)

FrequencyMatrix constructor initializes the matrix values to 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	the intial psuedo-counts (un-logged).
logged	bool that specifies if the table is to be stored logged.

Definition at line 143 of file frequencymatrix.h.

3.10.3 Member Function Documentation

3.10.3.1 template<class T > size_t FrequencyMatrix< T >::argmax (size_t *i*) const

A member function that finds and returns the argmax (index of mode) of the given distribution.

Parameters

i	the distribution (row).

Returns

The argmax of the distribution.

Definition at line 217 of file frequencymatrix.h.

3.10.3.2 template<class T > void FrequencyMatrix< T >::fix ( )

A member function that normalizes and "locks" the matrix values so that no changes can be made.

This allows for faster future lookups, but is irrevocable.

Definition at line 232 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).

Does nothing if _fixed is true.

Parameters

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

Definition at line 189 of file frequencymatrix.h.

3.10.3.4 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 171 of file frequencymatrix.h.

3.10.3.5 template<class T > T FrequencyMatrix< T >::operator() (size_t *k*, bool *normalized* = `true`) const

An accessor for the frequency at a given position in the flattened matrix (logged if table is logged).

Parameters

k	the array position.
normalized	a bool specifying whether or not the frequency should be normalized.

Returns

The frequency at the given position in the flattened matrix (logged if table is logged).

Definition at line 166 of file frequencymatrix.h.

3.10.3.6 template<class T > T FrequencyMatrix< T >::operator() (size_t *i*, size_t *j*, bool *normalized* = `true`) const

An accessor for the frequency at 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

The frequency of the given value in the given distribution (logged if table is logged).

Definition at line 153 of file frequencymatrix.h.

3.10.3.7 template<class T > void FrequencyMatrix< T >::set_logged (bool *logged*)

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

Does nothing if _fixed is true.

Parameters

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

Definition at line 194 of file frequencymatrix.h.

3.10.3.8 template<class T> T FrequencyMatrix< T >::sum (size_t *i*) const [inline]

An accessor for the row sum (normalizer), (logged if table is logged).

Parameters

i	the distribution (row).

Returns

The sum (normalizer) for the given distribution (logged if table is (logged).

Definition at line 114 of file frequencymatrix.h.

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

- src/frequencymatrix.h

3.11 HaplotypeHandler Class Reference

The HaplotypeHandler class keeps track of sets of transcripts from different chromosomes.

`#include <targets.h>`

Public Member Functions

- HaplotypeHandler (std::vector< Target * > targets, double alpha)

  Constructor for the HaplotypeHandler.

- double get_mass (const Target *targ, bool with_pseudo)

  Gets the current relative mass of the given target within the set.

- void update_mass (const Target *targ, const std::string &frag_name, double align_-likelihood, double mass)

  Buffers the mass and likelihood assigned to the given fragment for the given target.

3.11.1 Detailed Description

The HaplotypeHandler class keeps track of sets of transcripts from different chromosomes. For these sets, a combined likelihood is computed relative to targets not in the set, but abundance within the set is calculated based only on fragments with likelihoods that differ within the set.

Due to how the fragments are processed in the main thread, likelihoods and global assigned masses are stored when first computed and then split within the set (committed) only when it is known that processing of the fragment is complete.

Author

Adam Roberts

Date

2013 Artistic License 2.0

Definition at line 446 of file targets.h.

3.11.2 Constructor & Destructor Documentation

3.11.2.1 HaplotypeHandler::HaplotypeHandler (std::vector< Target * > *targets*, double *alpha*)

Constructor for the HaplotypeHandler.

Provides all targets a shared pointer to this handler, effectively passing ownership to them.

Definition at line 232 of file targets.cpp.

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

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

3.12 HitParams Struct Reference

The HitParams struct stores likelihood information for a single hit of a fragment.

`#include <fragments.h>`

Public Attributes

- double **align_likelihood**
- double **full_likelihood**
- double **posterior**

3.12.1 Detailed Description

The HitParams struct stores likelihood information for a single hit of a fragment.

Definition at line 129 of file fragments.h.

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

- src/fragments.h

3.13 Indel Struct Reference

The Indel struct stores the information for a single insertion or deletion.

`#include <fragments.h>`

Public Member Functions

- Indel (size_t p, size_t l)

  Indel constructor.

Public Attributes

- size_t pos

  A public size_t for the position of the Indel in the read.

- size_t len

  A public size_t for the length of the Indel in the read.

3.13.1 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 42 of file fragments.h.

3.13.2 Member Data Documentation

3.13.2.1 size_t Indel::pos

A public size_t for the position of the Indel in the read.

0-based.

Definition at line 46 of file fragments.h.

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

- src/fragments.h

3.14 LengthDistribution Class Reference

lengthdistribution.h express

```
#include <lengthdistribution.h>
```

Public Member Functions

- LengthDistribution (double alpha, size_t max_val, size_t prior_mu, size_t prior_sigma, size_t kernel_n, double kernel_p, size_t bin_size=1)

  LengthDistribution Constructor.

- LengthDistribution (std::string param_file_name, std::string length_type)

  A second constructor that loads the distribution from a parameter file.

- size_t max_val () const

  An accessor for the maximum allowed length.

- size_t min_val () const

  An accessor for the minimum observed length (1 initially).

- double mean () const

  An accessor for the mean length in the distribution.

- void add_val (size_t len, double mass)

  A member function that updates the distribution based on a new length observation.

- double pmf (size_t len) const

  An accessor for the (logged) probability of a given length.

- double cmf (size_t len) const

  A member function that returns a (logged) cumulative mass for a given length.

- std::vector< double > cmf () const

  A member function that returns a vector containing the (logged) cumulative mass function ∗for the bins∗.

- double tot_mass () const

  An accessor for the (logged) observation mass (including pseudo-counts).

- std::string to_string () const

  A member function that returns a string containing the current distribution.

- void append_output (std::ofstream &outfile, std::string length_type) const

  A member function that appends the LengthDistribution parameters to the end of the given file.

3.14.1 Detailed Description

lengthdistribution.h express Created by Adam Roberts on 1/30/13. Copyright 2013 Adam Roberts. All rights reserved. The LengthDistribution class keeps track of the observed length distribution. It is initialized with a Gaussian prior with parameters specified by the arguments to the constructor. An argument-specified binomial kernel is then added for each observation. All mass values and probabilities are stored and returned in log space (except in to_string).

Definition at line 22 of file lengthdistribution.h.

3.14.2 Constructor & Destructor Documentation

3.14.2.1 LengthDistribution::LengthDistribution (double *alpha*, size_t *max_val*, size_t *prior_mu*, size_t *prior_sigma*, size_t *kernel_n*, double *kernel_p*, size_t *bin_size* = `1`)

LengthDistribution Constructor.

Parameters

alpha	double that sets the average pseudo-counts (logged).
max_val	an integer that sets the maximum allowable length.
prior_mu	a size_t for the mean of the prior gaussian distribution. If 0, a uniform distribution is used instead.
prior_sigma	a size_t for the standard deviation of the prior gaussian distribution.
kernel_n	a size_t specifying the number of trials in the kernel binomial distribution. Must be odd.
kernel_p	a double specifying the success probability for the kernel binomial distribution.
bin_size	a size_t specifying the size of bins to use internally to reduce the number of parameters in the distribution.

Definition at line 20 of file lengthdistribution.cpp.

3.14.2.2 LengthDistribution::LengthDistribution (std::string *param_file_name*, std::string *length_type*)

A second constructor that loads the distribution from a parameter file.

Note that the values should not be modified (add_val should not be called) after using this constructor. The bin size is set to 1.

Parameters

param_file_name	a string specifying the path to the parameter file.
length_type	a string specifying the type of length distribution to be matched in the parameter file.

3.14.3 Member Function Documentation

3.14.3.1 void LengthDistribution::add_val (size_t *len*, double *mass*)

A member function that updates the distribution based on a new length observation.

Parameters

len	an integer for the observed length.
mass	a double for the mass (logged) to add.

Definition at line 114 of file lengthdistribution.cpp.

3.14.3.2 void LengthDistribution::append_output (std::ofstream & *outfile*, std::string *length_type*) const

A member function that appends the LengthDistribution parameters to the end of the given file.

Parameters

outfile	the file to append to.
length_type	a string specifying the type of length the distribution is of (ie. "Fragment" or "Target") to be included in the header.

3.14.3.3 double LengthDistribution::cmf (size_t *len*) const

A member function that returns a (logged) cumulative mass for a given length.

Parameters

len	an integer for the length to return the cmf value of.

Returns

(Logged) cmf value of length.

Definition at line 148 of file lengthdistribution.cpp.

3.14.3.4 vector< double > LengthDistribution::cmf ( ) const

A member function that returns a vector containing the (logged) cumulative mass function ∗for the bins∗.

Returns

(Logged) cmf of bins.

Definition at line 158 of file lengthdistribution.cpp.

3.14.3.5 size_t LengthDistribution::max_val ( ) const

An accessor for the maximum allowed length.

Returns

Max allowed length.

Definition at line 103 of file lengthdistribution.cpp.

3.14.3.6 double LengthDistribution::mean ( ) const

An accessor for the mean length in the distribution.

Returns

Mean observed length.

Definition at line 174 of file lengthdistribution.cpp.

3.14.3.7 size_t LengthDistribution::min_val ( ) const

An accessor for the minimum observed length (1 initially).

Returns

Minimum observed length.

Definition at line 107 of file lengthdistribution.cpp.

3.14.3.8 double LengthDistribution::pmf (size_t *len*) const

An accessor for the (logged) probability of a given length.

Parameters

len	an integer for the length to return the probability of.

Returns

(logged) probability of observing the given length.

Definition at line 140 of file lengthdistribution.cpp.

3.14.3.9 string LengthDistribution::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 178 of file lengthdistribution.cpp.

3.14.3.10 double LengthDistribution::tot_mass ( ) const

An accessor for the (logged) observation mass (including pseudo-counts).

Returns

Total observation mass.

Definition at line 170 of file lengthdistribution.cpp.

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

- src/lengthdistribution.h
- src/lengthdistribution.cpp

3.15 Librarian Class Reference

The Librarian class keeps track of the different library objects for a run.

`#include <library.h>`

Public Member Functions

- Librarian (size_t num_libs)

  Librarian Constructor.

- Library & operator[ ] (size_t i)

  An accessor for the Library struct at a given index.

- const Library & curr_lib () const

  An accessor for the Library struct associated with the library currently being processed.

- void set_curr (size_t i)

  A mutator of the index of the library currently being processed.

- size_t size () const

  An accessor for the number of Library structs.

3.15.1 Detailed Description

The Librarian class keeps track of the different library objects for a run.

Author

Adam Roberts

Date

2012 Artistic License 2.0

Definition at line 75 of file library.h.

3.15.2 Constructor & Destructor Documentation

3.15.2.1 Librarian::Librarian (size_t *num_libs*) [inline]

Librarian Constructor.

Parameters

num_libs	a size_t for the number of libraries to be processed in the run.

Definition at line 92 of file library.h.

3.15.3 Member Function Documentation

3.15.3.1 const Library& Librarian::curr_lib ( ) const [inline]

An accessor for the Library struct associated with the library currently being processed.

Returned value does not outlive this.

Returns

The Library struct indexed by _curr.

Definition at line 108 of file library.h.

3.15.3.2 Library& Librarian::operator[] (size_t *i*) [inline]

An accessor for the Library struct at a given index.

Returned value does not outlive this.

Parameters

i	a size_t indexing the requested Library struct.

Returns

The Library struct at the given index.

Definition at line 99 of file library.h.

3.15.3.3 void Librarian::set_curr (size_t *i*) [inline]

A mutator of the index of the library currently being processed.

Parameters

i	a size_t to set the index of the current Library struct to.

Definition at line 113 of file library.h.

3.15.3.4 size_t Librarian::size ( ) const [inline]

An accessor for the number of Library structs.

This should be equal to the number of libraries to be processed in the run.

Returns

The number of Library structs.

Definition at line 122 of file library.h.

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

- src/library.h

3.16 Library Struct Reference

library.h express

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

Public Member Functions

- Library ()

  Library constructor sets initial values for parameters.

Public Attributes

- std::string in_file_name

  Path to the input file.

- std::string out_file_name

  Path to the out file.

- boost::shared_ptr< MapParser > map_parser

  A pointer to the MapParser for parsing the input alignment file for this library.

- boost::shared_ptr< LengthDistribution > fld

  A pointer to the fragment length distribution object for this library.

- boost::shared_ptr< MismatchTable > mismatch_table

  A pointer to the MismatchTable containing the learned error distribution for this library.

- boost::shared_ptr< BiasBoss > bias_table

  A pointer to the BiasBoss containing the learned bias distribution for this library.

- boost::shared_ptr< TargetTable > targ_table

  A pointer to the TargetTable containing the target parameters (abundance, effective length) for this library.

- size_t n

  The number of the next read to be processed (starting at 1).

- double mass_n

  The mass of the next read to be processed (logged).

3.16.1 Detailed Description

library.h express Created by Adam Roberts on 5/12/12. Copyright (c) 2012 Adam Roberts. All rights reserved. The Library struct holds pointers to the global parameter tables for a set of reads from the same library preparation.

Author

Adam Roberts

Date

2012 Artistic License 2.0

Definition at line 22 of file library.h.

3.16.2 Member Data Documentation

3.16.2.1 boost::shared_ptr<BiasBoss> Library::bias_table

A pointer to the BiasBoss containing the learned bias distribution for this library.

(optional)

Definition at line 49 of file library.h.

3.16.2.2 std::string Library::in_file_name

Path to the input file.

Empty if streamed.

Definition at line 26 of file library.h.

3.16.2.3 std::string Library::out_file_name

Path to the out file.

Empty if alignments are not to be output.

Definition at line 30 of file library.h.

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

- src/library.h

3.17 Logger Class Reference

logger.cpp express

```
#include <logger.h>
```

Public Member Functions

- void **info_out** (std::ostream *out)
- void **warn_out** (std::ostream *out)
- void **severe_out** (std::ostream *out)
- void **info** (const char *msg,...) const
- void **warn** (const char *msg,...) const
- void **severe** (const char *msg,...) const

3.17.1 Detailed Description

logger.cpp express Created by Adam Roberts on 6/22/13. Copyright 2013 Adam Roberts. All rights reserved.

Definition at line 20 of file logger.h.

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

- src/logger.h

3.18 MapParser Class Reference

The MapParser class is meant to be run as a separate thread from the main processing.

`#include <mapparser.h>`

Public Member Functions

- MapParser (Library *lib, bool write_active)

  MapParser constructor determines what format the input is in and initializes the correct parser and writer (if appropriate).

- void threaded_parse (ParseThreadSafety *thread_safety, size_t stop_at=0, size_t num_neighbors=0)

  A member function that drives the parse thread.

- const TransIndex & targ_index ()

  An accessor for the target name to index map.

- const TransIndex & targ_lengths ()

  An accessor for the target-to-length map.

- void write_active (bool b)

  A mutator for the write-active status of the parser.

- void reset_reader ()

  A member function that resets the input parser.

3.18.1 Detailed Description

The MapParser class is meant to be run 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 308 of file mapparser.h.

3.18.2 Constructor & Destructor Documentation

3.18.2.1 MapParser::MapParser (Library * *lib*, bool *write_active*)

MapParser constructor determines what format the input is in and initializes the correct parser and writer (if appropriate).

Parameters

lib	pointer to variables associated with the input, including file path.
write_active	bool to initialize _write_active.

Definition at line 96 of file mapparser.cpp.

3.18.3 Member Function Documentation

3.18.3.1 const TransIndex& MapParser::targ_index ( ) [inline]

An accessor for the target name to index map.

Returns a reference that does not outlive this.

Returns

Reference to the target-to-index map.

Definition at line 359 of file mapparser.h.

3.18.3.2 const TransIndex& MapParser::targ_lengths ( ) [inline]

An accessor for the target-to-length map.

Returns a reference that does not outlive this.

Returns

Reference to the target-to-length map.

Definition at line 365 of file mapparser.h.

3.18.3.3 void MapParser::threaded_parse (ParseThreadSafety * *thread_safety*, size_t *stop_at* = 0, size_t *num_neighbors* = 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 a queue in the ParseThreadSafety struct. After processing, the Fragment returns on a different in queue, and is written to the output map file (depending on settings) and deleted.

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, default).
num_neighbors	experimental.

Definition at line 149 of file mapparser.cpp.

3.18.3.4 void MapParser::write_active (bool *b*) [inline]

A mutator for the write-active status of the parser.

This specifies whether or not the alignments (sampled or with probs) should be ouptut.

Parameters

b	updated write-active status

Definition at line 371 of file mapparser.h.

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

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

3.19 MarkovModel Class Reference

The MarkovModel class is used to store transition probabilities of a Markov chain based on a nucleotide Sequence, which itself can be probabilistic.

`#include <markovmodel.h>`

Public Member Functions

- MarkovModel (size_t order, size_t window_size, size_t num_pos, double alpha)

  MarkovModel Constructor.

- double transition_prob (size_t p, size_t cond, size_t curr) const

  Accessor for the probability of transitioning from cond to curr at node p.

- double seq_prob (const Sequence &seq, int left) const

  Computes the probability of the sequence beginning at left of size _window_size using the parameters of the Markov model.

- void update (const Sequence &seq, int left, double mass)

  Increments the parameters associated with the sequence beginning at left of size _window_size by the (logged) mass.

- void update (size_t p, size_t i, size_t j, double mass)

  Increments the specified transition by the given mass.

- size_t get_indices (const Sequence &seq, int left, std::vector< char > &indices)

  A member function that computes and returns the parameter table indices used to compute and update the likelihood for the given sequence at the window started at the given position.

- std::vector< char > get_indices (const Sequence &seq)

  A member function that computes and returns the parameter table indices used to fast_learn (see below) the likelihood for the given sequence.

- double marginal_prob (size_t w, size_t nuc) const

  Computes the marginal probability of transitioning to the given nucleotide at position w in the model.

- void fast_learn (const Sequence &seq, double mass, const std::vector< double > &fl_cmf)

  Slides a window along the given sequence, incrementing the highest order transition paramaters by the given mass multiplied by the probability of observing a fragment at that distance from the end.

- void calc_marginals ()

After learning the highest order transitions with fast_learn, this method fills in the lower-order transitions.

3.19.1 Detailed Description

The MarkovModel class is used to store transition probabilities of a Markov chain based on a nucleotide Sequence, which itself can be probabilistic. The The probabilities can be updated based on a sequence window and can be initialized by sliding a window over a sequence. Nucleotides are represented by size_t values according to the NUCS array in main.h.

Author

Adam Roberts

Date

2012 Artistic License 2.0

Definition at line 28 of file markovmodel.h.

3.19.2 Constructor & Destructor Documentation

3.19.2.1 MarkovModel::MarkovModel (size_t *order*, size_t *window_size*, size_t *num_pos*, double *alpha*)

MarkovModel Constructor.

Parameters

order	the order of the model.
window_size	the size of the sequence window to calculate probabilities for.
num_pos	the number of nodes in the model.
alpha	the initial pseudo-counts (non-logged).

Definition at line 16 of file markovmodel.cpp.

3.19.3 Member Function Documentation

3.19.3.1 void MarkovModel::fast_learn (const Sequence & *seq*, double *mass*, const std::vector< double > & *fl_cmf*)

Slides a window along the given sequence, incrementing the highest order transition paramaters by the given mass multiplied by the probability of observing a fragment at that distance from the end.

Parameters

seq	the sequence to slide the window along.
mass	the amount to increment the parameters by.
fl_cmf	the fragment length CMF to determine the probability of observing fragment starts at different positions in the sequence.

Definition at line 113 of file markovmodel.cpp.

3.19.3.2 size_t MarkovModel::get_indices (const Sequence & *seq*, int *left*, std::vector< char > & *indices*)

A member function that computes and returns the parameter table indices used to compute and update the likelihood for the given sequence at the window started at the given position.

Negative values indicate that the position is not used in the likelihood computation.

Parameters

seq	the sequence to find the indices for.
left	the position where the window should begin in the sequence.
indices	a vector to fill with indices into the parameter tables that would be used in a likelihood calculation.

Returns

The starting position in the sequence to align with the first index.

3.19.3.3 vector< char > MarkovModel::get_indices (const Sequence & *seq*)

A member function that computes and returns the parameter table indices used to fast_-learn (see below) the likelihood for the given sequence.

Negative values indicate that the position is not used in the fast_learn computation.

Parameters

seq	the sequence to find the indices for.

Returns

A vector of indices into the parameter tables that would be used in a fast_learn.

Definition at line 92 of file markovmodel.cpp.

3.19.3.4 double MarkovModel::marginal_prob (size_t *w*, size_t *nuc*) const

Computes the marginal probability of transitioning to the given nucleotide at position w in the model.

Parameters

w	the position (node) in the model.
nuc	the nucleotide to calculate the marginal transition probability to.

Returns

The marginal probability of transitioning to nuc at position w.

Definition at line 204 of file markovmodel.cpp.

3.19.3.5 double MarkovModel::seq_prob (const Sequence & *seq*, int *left*) const

Computes the probability of the sequence beginning at left of size _window_size using the parameters of the Markov model.

Parameters

seq	the sequence from which to extract the window.
left	the leftmost point in the sequence window.

Returns

The probability of the sequence based on the model parameters.

Definition at line 163 of file markovmodel.cpp.

3.19.3.6 double MarkovModel::transition_prob (size_t *p*, size_t *cond*, size_t *curr*) const

Accessor for the probability of transitioning from cond to curr at node p.

Parameters

p	the node to get the transition probability from.
cond	the index of the previous state.
curr	the index of the state being transitioned to.

Returns

The probability of transitioning from cond to curr at node p.

Definition at line 158 of file markovmodel.cpp.

3.19.3.7 void MarkovModel::update (const Sequence & *seq*, int *left*, double *mass*)

Increments the parameters associated with the sequence beginning at left of size _window_size by the (logged) mass.

Parameters

seq	the sequence from which to extract the window.
left	the leftmost point in the sequence window.
mass	the amount to increment the parameters by (logged).

Definition at line 57 of file markovmodel.cpp.

3.19.3.8 void MarkovModel::update (size_t *p*, size_t *i*, size_t *j*, double *mass*)

Increments the specified transition by the given mass.

Parameters

p	the position in the chain to increment.
i	the index of the previous state.
j	the index of the transitioned state.
mass	the amount to increment by (logged).

Definition at line 88 of file markovmodel.cpp.

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

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

3.20 MismatchTable Class Reference

The MismatchTable class is used to store and update mismatch and indel (error) parameters using a first-order Markov model based on nucleotide and position in a read.

`#include <mismatchmodel.h>`

Public Member Functions

- MismatchTable (double alpha)

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

- MismatchTable (std::string param_file_name)

  A second constructor that loads the distribution from a parameter file.

- void activate (bool active=true)

  Mutator to set the _active member variable to allow for log_likelihood calculations.

- void get_indices (const FragHit &f, std::vector< char > &left_indices, std::vector< char > &left_seq, std::vector< char > &left_ref, std::vector< char > &right_indices, std::vector< char > &right_seq, std::vector< char > &right_ref) const

A member function that computes and returns the parameter table indices used to compute and update the likelihood based on the given FragHit.

- double log_likelihood (const FragHit &f) const

  A member function that returns the log likelihood of mismatches and indels in the mapping given the current error model parematers.

- void update (const FragHit &, double p, double mass)

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

- void fix ()

  Freezes the parameters to allow for faster computation after burn out.

- void append_output (std::ofstream &outfile) const

  A member function that appends the final model parameters in tab-separated format to the given file.

3.20.1 Detailed Description

The MismatchTable class is used to store and update mismatch and indel (error) parameters using a first-order Markov model based on nucleotide and position in a read. Also computes likelihoods of mismatches and indels in given fragment mappings. When the target sequences are proabilistic, this class is responsible for updating those parameters. All values are stored and returned in log space.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 29 of file mismatchmodel.h.

3.20.2 Constructor & Destructor Documentation

3.20.2.1 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.

3.20.2.2 MismatchTable::MismatchTable (std::string *param_file_name*)

A second constructor that loads the distribution from a parameter file.

Note that the values should not be modified after using this constructor.

Parameters

param_file_name	a string specifying the path to the parameter file.

3.20.3 Member Function Documentation

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

Mutator to set the _active member variable to allow for log_likelihood calculations.

Used to skip calculations before burn-in completes.

Parameters

active	a boolean specifying whether to activate (true) or deactivate (false)

Definition at line 79 of file mismatchmodel.h.

3.20.3.2 void MismatchTable::append_output (std::ofstream & *outfile*) const

A member function that appends the final model parameters in tab-separated format to the given file.

The output 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 119 of file biascorrection.cpp.

3.20.3.3 void MismatchTable::fix ( )

Freezes the parameters to allow for faster computation after burn out.

Cannot be undone.

Definition at line 567 of file mismatchmodel.cpp.

3.20.3.4 void MismatchTable::get_indices (const FragHit & *f*, std::vector< char > & *left_indices*, std::vector< char > & *left_seq*, std::vector< char > & *left_ref*, std::vector< char > & *right_indices*, std::vector< char > & *right_seq*, std::vector< char > & *right_ref*) const

A member function that computes and returns the parameter table indices used to compute and update the likelihood based on the given FragHit.

Sequence values are compressed to 2 bits per nucleotide.

Parameters

f	the FragHit to find the table indices for.
left_indices	vector to store the left read's mismatched positions.
left_seq	vector to store the left read's mismatched nucleotides (compressed).
left_ref	vector to store the left read's reference nucleotides (compressed) at mismatch positions.
right_indices	vector to store the right read's mismatched positions.
right_seq	vector to store the rightt read's mismatched nucleotides (compressed).
right_ref	vector to store the right read's reference nucleotides (compressed) at mismatch positions.

Definition at line 159 of file mismatchmodel.cpp.

3.20.3.5 double MismatchTable::log_likelihood (const FragHit & *f*) const

A member function that returns the log likelihood of mismatches and indels in the mapping given the current error model paremeters.

Returns 0 if _active is false.

Parameters

f	the fragment mapping to calculate the log likelihood for.

Returns

The log likelihood of the mapping based on mismatches and indels.

Definition at line 264 of file mismatchmodel.cpp.

3.20.3.6 void MismatchTable::update (const FragHit & *f*, double *p*, double *mass*)

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

Also updates the sequence parameters if they are probabilistic and active_ is true.

Parameters

f	the fragment mapping.

p	the logged posterior probablity of the alignment.
mass	the logged mass of the fragment.

Definition at line 393 of file mismatchmodel.cpp.

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

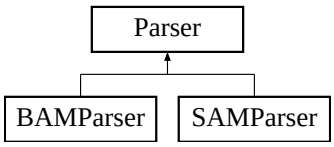
- src/mismatchmodel.h
- src/biascorrection.cpp
- src/lengthdistribution.cpp
- src/mismatchmodel.cpp

3.21 Parser Class Reference

The Parser class is an abstract class for implementing a SAMParser or BAMParser.

`#include <mapparser.h>`

Inheritance diagram for Parser:



Public Member Functions

- virtual ~Parser ()

  Dummy destructor.

- virtual const std::string header () const =0

  An accessor for the SAM header string.

- const TransIndex & targ_index () const

  An accessor for the target name to index map.

- const TransIndex & targ_lengths () const

  An accessor for the target-to-length map.

- virtual bool next_fragment (Fragment &f)=0

  A member function that loads all mappings of the next fragment into the given Fragment object.

- virtual void reset ()=0

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

Protected Attributes

- TransIndex _targ_index

  The private target-to-index map.

- TransIndex _targ_lengths

  The private target-to-length map.

- ReadHit * _read_buff

  A private pointer to the current/last read mapping being parsed.

3.21.1 Detailed Description

The Parser class is an abstract class for implementing 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 39 of file mapparser.h.

3.21.2 Member Function Documentation

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

An accessor for the SAM header string.

Returns

The SAM header string.

Implemented in BAMParser, and SAMParser.

3.21.2.2 virtual bool Parser::next_fragment (Fragment & *f*) [pure virtual]

A member function that loads all mappings of the next fragment into the given Fragment object.

Parameters

f	the empty Fragment to add mappings to.

Returns

True iff more reads remain in the SAM/BAM file/stream.

Implemented in BAMParser, and SAMParser.

3.21.2.3 const TransIndex& Parser::targ_index ( ) const [inline]

An accessor for the target name to index map.

Returns a reference that does not outlive this.

Returns

Reference to the target-to-index map.

Definition at line 69 of file mapparser.h.

3.21.2.4 const TransIndex& Parser::targ_lengths ( ) const [inline]

An accessor for the target-to-length map.

Returns a reference that does not outlive this.

Returns

Reference to the target-to-length map.

Definition at line 75 of file mapparser.h.

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

- src/mapparser.h

3.22 ParseThreadSafety Struct Reference

The ParseThreadSafety struct stores objects to allow for parsing to safely occur on a separate thread from processing.

`#include <threadsafety.h>`

Public Member Functions

- ParseThreadSafety (size_t q_size)

  PraseThreadSafety constructor intializes queues to the given size.

Public Attributes

- ThreadSafeFragQueue proc_in

  A public ThreadSafeFragQueue of pointers to Fragments that have been parsed but not pre-processed.

- ThreadSafeFragQueue proc_on

  A public ThreadSafeFragQueue of pointers to Fragments that have been pre-processed but not processed.

- ThreadSafeFragQueue proc_out

  A public ThreadSafeFragQueue of pointers to Fragments that have been processed but not post-processed.

3.22.1 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 85 of file threadsafety.h.

3.22.2 Constructor & Destructor Documentation

3.22.2.1 ParseThreadSafety::ParseThreadSafety (size_t *q_size*) [inline]

PraseThreadSafety constructor intializes queues to the given size.

Parameters

q_size	the maximum size for the ThreadSafeFragQueues.

Definition at line 105 of file threadsafety.h.

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

- src/threadsafety.h

3.23 ReadHit Struct Reference

The ReadHit struct stores information for a single read alignment.

`#include <fragments.h>`

Public Attributes

- std::string name

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

- bool first

  A public bool specifying if this read was sequenced first according to the SAM flag.

- bool reversed

  A public bool specifying if this read was reverse complemented in its alignment according to the SAM flag.

- size_t targ_id

  A public TargID for the target mapped to.

- size_t left

  A public size_t containing the 0-based leftmost coordinate mapped to in the target.

- size_t right

  A public size_t containing the position following the 0-based rightmost coordinate mapped to in the target.

- SequenceFwd seq

  The read sequence.

- std::vector< Indel > inserts

  A public vector of Indel objects storing all insertions to the reference in the read.

- std::vector< Indel > deletes

  A public vector of Indel objects storing all insertions to the reference in the read.

- BamTools::BamAlignment bam

  A public BamAlignment object storing the raw alignment information from BamTools for the read.

- std::string sam

  A public string storing the raw alignment information from for the read.

- int mate_l

  A public int containing the left position for the mate of the read.

3.23.1 Detailed Description

The ReadHit struct stores information for a single read alignment.

Author

Adam Roberts

Date

2012 Artistic License 2.0

Definition at line 63 of file fragments.h.

3.23.2 Member Data Documentation

3.23.2.1 BamTools::BamAlignment ReadHit::bam

A public BamAlignment object storing the raw alignment information from BamTools for the read.

Only valid if BAM file is input.

Definition at line 111 of file fragments.h.

3.23.2.2 std::vector<Indel> ReadHit::deletes

A public vector of Indel objects storing all insertions to the reference in the read.

Deletions are stored in read order.

Definition at line 106 of file fragments.h.

3.23.2.3 std::vector<Indel> ReadHit::inserts

A public vector of Indel objects storing all insertions to the reference in the read.

Insertions are stored in read order.

Definition at line 101 of file fragments.h.

3.23.2.4 int ReadHit::mate_l

A public int containing the left position for the mate of the read.

-1 if single-end fragment. This is temporarily used to help find the mate but is not used after.

Definition at line 122 of file fragments.h.

3.23.2.5 bool ReadHit::reversed

A public bool specifying if this read was reverse complemented in its alignment according to the SAM flag.

This would also imply that the read is the left end of the fragment.

Definition at line 78 of file fragments.h.

3.23.2.6 std::string ReadHit::sam

A public string storing the raw alignment information from for the read.

Only valid if SAM file is input.

Definition at line 116 of file fragments.h.

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

- src/fragments.h

3.24 Result Struct Reference

Public Member Functions

- void **set_zeros** ()

Public Attributes

- double **fpkm**
- double **fpkm_std_dev**
- double **fpkm_lo**
- double **fpkm_hi**
- double **count_alpha**
- double **count_beta**
- double **est_counts**
- double **eff_len**
- double **eff_counts**
- double **cpb**

3.24.1 Detailed Description

Definition at line 527 of file targets.cpp.

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

- src/targets.cpp

3.25 RobertsFilter Class Reference

The RobertsFilter class implements a datastructure to test for repeats of a key with high probability, when repeats are most likely to be nearby.

`#include <robertsfilter.h>`

Public Member Functions

- RobertsFilter (size_t local_size=DEFAULT_LOC_SIZE, size_t global_size=DEFAULT_GLOB_SIZE)

  RobertsFilter constructor sets the size of the sets.

- bool test_and_push (const std::string &key)

  A member function that tests for membership of the key in either set.

3.25.1 Detailed Description

The RobertsFilter class implements a datastructure to test for repeats of a key with high probability, when repeats are most likely to be nearby. Recently observed keys are stored in a local set for a certain number of observations (set by local_size). After this number of observations, it is removed from the local set, and placed in the global set, displacing a random element of this set. To be used when the full set cannot be stored in memory.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 30 of file robertsfilter.h.

3.25.2 Constructor & Destructor Documentation

3.25.2.1 RobertsFilter::RobertsFilter (size_t *local_size* = DEFAULT_LOC_SIZE, size_t *global_size* = DEFAULT_GLOB_SIZE)

RobertsFilter constructor sets the size of the sets.

Parameters

local_size	the maximum number of keys to store in the local set.
global_size	the maximum number of keys to store in the global set.

Definition at line 13 of file robertsfilter.cpp.

3.25.3 Member Function Documentation

3.25.3.1 bool RobertsFilter::test_and_push (const std::string & *key*)

A member function that tests for membership of the key in either set.

If not found, the key is added to the local set, possibly pushing the oldest key in the local set into the global set, which may displace a global key.

Parameters

key	the key to be tested for and pushed into the local set.

Returns

True iff the key is in the local or global set.

Definition at line 19 of file robertsfilter.cpp.

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

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

3.26 RoundParams Struct Reference

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

`#include <targets.h>`

Public Member Functions

- RoundParams ()

  RoundParams constructor sets initial values for parameters.

Public Attributes

- double mass

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

- double ambig_mass

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

- double tot_ambig_mass

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

- double mass_var

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

- double var_sum

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

- boost::shared_ptr< HaplotypeHandler > **haplotype**

3.26.1 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 40 of file targets.h.

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

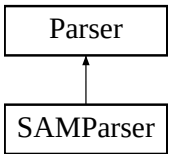
- src/targets.h

3.27 SAMParser Class Reference

The SAMParser class fills Fragment objects by parsing an input in SAM format.

`#include <mapparser.h>`

Inheritance diagram for SAMParser:



Public Member Functions

- SAMParser (std::istream *in)

  SAMParser constructor removes the header and parses the first line to start the first Fragment.

- const std::string header () const

  An accessor for the header string.

- bool next_fragment (Fragment &f)

  A member function that loads all mappings of the next fragment into the given Fragment object.

- void reset ()

  A member function that resets the parser and rewinds to the beginning of the SAM file.

3.27.1 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 174 of file mapparser.h.

3.27.2 Constructor & Destructor Documentation

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

SAMParser constructor removes the header and parses the first line to start the first Fragment.

Parameters

in	the input stream in SAM format, which may be a file or stdin.

Definition at line 355 of file mapparser.cpp.

3.27.3 Member Function Documentation

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

An accessor for the header string.

Returns

The header string.

Implements Parser.

Definition at line 203 of file mapparser.h.

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

A member function that loads all mappings of the next fragment into the given Fragment object.

Parameters

f	the empty Fragment to add mappings to.

Returns

True iff more reads remain in the SAM/BAM file/stream.

Implements Parser.

Definition at line 400 of file mapparser.cpp.

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

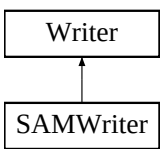
- src/mapparser.h
- src/mapparser.cpp

3.28 SAMWriter Class Reference

The SAMWriter class writes Fragment objects back to file in SAM format with per-mapping probabilistic assignments, or by sampling a single mapping based on assignment probabilities.

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

Inheritance diagram for SAMWriter:



Public Member Functions

- SAMWriter (std::ostream ∗out, bool sample)

  SAMWriter constructor stores a pointer to the output stream.

- ∼SAMWriter ()

  SAMWriter destructor flushes the output stream.

- void write_fragment (Fragment &f)

  A member function that writes the mappings to the output SAM file.

3.28.1 Detailed Description

The SAMWriter class writes Fragment objects back to file in SAM format with per-mapping probabilistic assignments, or by sampling a single mapping based on assignment probabilities.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 267 of file mapparser.h.

3.28.2 Constructor & Destructor Documentation

3.28.2.1 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 based on posteriors (true) or all output with their respective posterior probabilities (false).

Definition at line 587 of file mapparser.cpp.

3.28.3 Member Function Documentation

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

A member function that writes the mappings to the output SAM file.

If _sample is true, a only one alignment is output, otherwise all mappings are output along with their probabilities in the "XP" field.

Parameters

f	the processed Fragment to output alignments of.

Implements Writer.

Definition at line 595 of file mapparser.cpp.

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

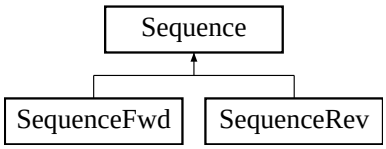
- src/mapparser.h
- src/mapparser.cpp

3.29 Sequence Class Reference

The Sequence class is an abstract class whose implmentations are used to store and access encoded nucleotide sequences.

`#include <sequence.h>`

Inheritance diagram for Sequence:



Public Member Functions

- Sequence ()

  Dummy Sequence constructor.

- virtual ~Sequence ()

  Dummy Sequence destructor.

- virtual void set (const std::string &seq, bool rev)=0

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

- virtual size_t operator[ ] (const size_t index) const =0

  An accessor for the encoded character at the given index.

- virtual size_t get_ref (const size_t index) const =0

  An accessor for the encoded reference character at the given index.

- virtual void update_est (const size_t index, const size_t nuc, float mass)=0

  A member function that updates the posterior nucleotide distribution if probabilistic.

- virtual void update_obs (const size_t index, const size_t nuc, float mass)=0

  A member function that updates the observed frequency distribution if probabilistic.

- virtual void update_exp (const size_t index, const size_t nuc, float mass)=0

  A member function that updates the expected frequency distribution if probabilistic.

- virtual float get_prob (const size_t index, const size_t nuc) const =0

  An accessor for the posterior nucleotide distribution (logged).

- virtual float get_obs (const size_t index, const size_t nuc) const =0

  An accessor for the observed nucleotide frequency (logged).

- virtual float get_exp (const size_t index, const size_t nuc) const =0

  An accessor for the expected nucleotide frequency (logged).

- virtual bool prob () const =0

  Accessor to determine if the sequence is probabilistic.

- virtual size_t length () const =0

  An accessor for the length of the encoded sequence.

- virtual bool empty () const =0

  Accessor to determine if the sequence has 0 length.

- virtual void calc_p_vals (std::vector< double > &p_vals) const =0

  A member function that calculates p-values based on the observed and expected nucleotide frequences for the sequence.

- std::string serialize ()

  A member function to serialize a string into an array of bytes, with each nucleotide represented by 2 bits.

3.29.1 Detailed Description

The Sequence class is an abstract class whose implmentations are used to store and access encoded nucleotide sequences. They also supports probabilistic sequences, meaning that each position stores a distribution over nucleotides.

Author

Adam Roberts

Date

2012 Artistic License 2.0

Definition at line 58 of file sequence.h.

3.29.2 Member Function Documentation

3.29.2.1 virtual void Sequence::calc_p_vals (std::vector< double > & *p_vals*) const [pure virtual]

A member function that calculates p-values based on the observed and expected nucleotide frequences for the sequence.

Experimental.

Parameters

p_vals	a reference to an empty vector of doubles to fill with p-values at each position.

Implemented in SequenceFwd, and SequenceRev.

3.29.2.2 virtual bool Sequence::empty ( ) const [pure virtual]

Accessor to determine if the sequence has 0 length.

Returns

True iff the sequence has 0 length.

Implemented in SequenceFwd, and SequenceRev.

3.29.2.3 virtual float Sequence::get_exp (const size_t *index*, const size_t *nuc*) const [pure virtual]

An accessor for the expected nucleotide frequency (logged).

Parameters

index	the index of the position to access.
nuc	the nucleotide to return the frequency of.

Returns

The logged expected frequency of the given nucleotide a the given position.

Implemented in SequenceFwd, and SequenceRev.

3.29.2.4 virtual float Sequence::get_obs (const size_t *index*, const size_t *nuc*) const [pure virtual]

An accessor for the observed nucleotide frequency (logged).

Parameters

index	the index of the position to access.
nuc	the nucleotide to return the frequency of.

Returns

The logged observed frequency of the given nucleotide a the given position.

Implemented in SequenceFwd, and SequenceRev.

3.29.2.5 virtual float Sequence::get_prob (const size_t *index*, const size_t *nuc*) const [pure virtual]

An accessor for the posterior nucleotide distribution (logged).

Parameters

index	the index of the position to access.
nuc	the nucleotide to return the probability of.

Returns

The logged posterior probability of the given nucleotide at the given position.

Implemented in SequenceFwd, and SequenceRev.

3.29.2.6 virtual size_t Sequence::get_ref (const size_t *index*) const [pure virtual]

An accessor for the encoded reference character at the given index.

May differ from the operator[] if the sequence is probabilistic.

Parameters

index	the index of the encoded reference character to return (assumed to be < _len)

Returns

The encoded reference character at the given index.

Implemented in SequenceFwd, and SequenceRev.

3.29.2.7 virtual size_t Sequence::length ( ) const [pure virtual]

An accessor for the length of the encoded sequence.

Returns

The length of the encoded sequence.

Implemented in SequenceFwd, and SequenceRev.

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

An accessor for the encoded character at the given index.

If the sequence is probabilistic, returns the mode. Otherwise, returns the reference.

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.

3.29.2.9 virtual bool Sequence::prob ( ) const [pure virtual]

Accessor to determine if the sequence is probabilistic.

Returns

True iff the sequence is probabilistic.

Implemented in SequenceFwd, and SequenceRev.

3.29.2.10 virtual void Sequence::set (const std::string & *seq*, bool *rev*) [pure virtual]

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.

Implemented in SequenceFwd, and SequenceRev.

3.29.2.11 virtual void Sequence::update_est (const size_t *index*, const size_t *nuc*, float *mass*) [pure virtual]

A member function that updates the posterior nucleotide distribution if probabilistic.

Parameters

index	the index of the position to update.
nuc	the nucleotide to increment the mass of.
mass	the amount to increment the nucleotide's mass by.

Implemented in SequenceFwd, and SequenceRev.

3.29.2.12 virtual void Sequence::update_exp (const size_t *index*, const size_t *nuc*, float *mass*) [pure virtual]

A member function that updates the expected frequency distribution if probabilistic.

Parameters

index	the index of the position to update.
nuc	the nucleotide to increment the mass of.
mass	the amount to increment the nucleotide's mass by.

Implemented in SequenceFwd, and SequenceRev.

3.29.2.13 virtual void Sequence::update_obs (const size_t *index*, const size_t *nuc*, float *mass*) [pure virtual]

A member function that updates the observed frequency distribution if probabilistic.

Parameters

index	the index of the position to update.
nuc	the nucleotide to increment the mass of.
mass	the amount to increment the nucleotide's mass by.

Implemented in SequenceFwd, and SequenceRev.

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

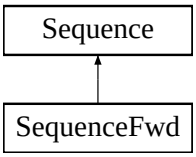
- src/sequence.h
- src/sequence.cpp

3.30 SequenceFwd Class Reference

The SequenceFwd class implements the Sequence abstract class for storing the forward sequence.

`#include <sequence.h>`

Inheritance diagram for SequenceFwd:



Public Member Functions

- SequenceFwd ()

  Dummy SequenceFwd constructor.

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

  SequenceFwd constructor encodes and stores the given nucleotide sequence.

- SequenceFwd (const SequenceFwd &other)

  SequenceFwd copy constructor.

- SequenceFwd & operator= (const SequenceFwd &other)

  SequenceFwd assignment constructor, copies the given SequenceFwd object.

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

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

- size_t operator[ ] (const size_t index) const

  An accessor for the encoded character at the given index.

- size_t get_ref (const size_t index) const

  An accessor for the encoded reference character at the given index.

- float get_exp (const size_t index, const size_t nuc) const

  An accessor for the expected nucleotide frequency (logged).

- float get_obs (const size_t index, const size_t nuc) const

  An accessor for the observed nucleotide frequency (logged).

- void update_est (const size_t index, const size_t nuc, float mass)

  A member function that updates the posterior nucleotide distribution if probabilistic.

- void update_obs (const size_t index, const size_t nuc, float mass)

  A member function that updates the observed frequency distribution if probabilistic.

- void update_exp (const size_t index, const size_t nuc, float mass)

  A member function that updates the expected frequency distribution if probabilistic.

- float get_prob (const size_t index, const size_t nuc) const

  An accessor for the posterior nucleotide distribution (logged).

- bool prob () const

  Accessor to determine if the sequence is probabilistic.

- bool empty () const

  Accessor to determine if the sequence has 0 length.

- size_t length () const

  An accessor for the length of the encoded sequence.

- void calc_p_vals (std::vector< double > &p_vals) const

  A member function that calculates p-values based on the observed and expected nucleotide frequences for the sequence.

3.30.1 Detailed Description

The SequenceFwd class implements the Sequence abstract class for storing the forward sequence. Documentation is only provided for methods not documented in the abstract Sequence class.

Author

Adam Roberts

Date

2012 Artistic License 2.0

Definition at line 177 of file sequence.h.

3.30.2 Constructor & Destructor Documentation

3.30.2.1 SequenceFwd::SequenceFwd (const std::string & *seq*, bool *rev*, bool *prob* = `false`)

SequenceFwd constructor encodes and stores the given nucleotide sequence.

Parameters

seq	the nucleotide sequence string to encode and store.
rev	a boolean if the sequence should be reverse complemented before encoding.

Definition at line 30 of file sequence.cpp.

3.30.2.2 SequenceFwd::SequenceFwd (const SequenceFwd & *other*)

SequenceFwd copy constructor.

Parameters

other	the SequenceFwd object to copy.

Definition at line 40 of file sequence.cpp.

3.30.3 Member Function Documentation

3.30.3.1 void SequenceFwd::calc_p_vals (std::vector< double > & *p_vals*) const [virtual]

A member function that calculates p-values based on the observed and expected nucleotide frequences for the sequence.

Experimental.

Parameters

p_vals	a reference to an empty vector of doubles to fill with p-values at each position.

Implements Sequence.

3.30.3.2 bool SequenceFwd::empty ( ) const [inline, virtual]

Accessor to determine if the sequence has 0 length.

Returns

True iff the sequence has 0 length.

Implements Sequence.

Definition at line 242 of file sequence.h.

3.30.3.3 float SequenceFwd::get_exp (const size_t *index*, const size_t *nuc*) const [virtual]

An accessor for the expected nucleotide frequency (logged).

Parameters

index	the index of the position to access.
nuc	the nucleotide to return the frequency of.

Returns

The logged expected frequency of the given nucleotide a the given position.

Implements Sequence.

Definition at line 99 of file sequence.cpp.

3.30.3.4 float SequenceFwd::get_obs (const size_t *index*, const size_t *nuc*) const [virtual]

An accessor for the observed nucleotide frequency (logged).

Parameters

index	the index of the position to access.
nuc	the nucleotide to return the frequency of.

Returns

The logged observed frequency of the given nucleotide a the given position.

Implements Sequence.

Definition at line 94 of file sequence.cpp.

3.30.3.5 float SequenceFwd::get_prob (const size_t *index*, const size_t *nuc*) const [virtual]

An accessor for the posterior nucleotide distribution (logged).

Parameters

index	the index of the position to access.
nuc	the nucleotide to return the probability of.

Returns

The logged posterior probability of the given nucleotide at the given position.

Implements Sequence.

Definition at line 89 of file sequence.cpp.

3.30.3.6 size_t SequenceFwd::get_ref (const size_t *index*) const [virtual]

An accessor for the encoded reference character at the given index.

May differ from the operator[] if the sequence is probabilistic.

Parameters

index	the index of the encoded reference character to return (assumed to be < _len)

Returns

The encoded reference character at the given index.

Implements Sequence.

Definition at line 83 of file sequence.cpp.

3.30.3.7 size_t SequenceFwd::length ( ) const [inline, virtual]

An accessor for the length of the encoded sequence.

Returns

The length of the encoded sequence.

Implements Sequence.

Definition at line 243 of file sequence.h.

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

SequenceFwd assignment constructor, copies the given SequenceFwd object.

Parameters

other	the Sequence object to copy.

Definition at line 50 of file sequence.cpp.

3.30.3.9 size_t SequenceFwd::operator[] (const size_t *index*) const [virtual]

An accessor for the encoded character at the given index.

If the sequence is probabilistic, returns the mode. Otherwise, returns the reference.

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 75 of file sequence.cpp.

3.30.3.10 bool SequenceFwd::prob ( ) const [inline, virtual]

Accessor to determine if the sequence is probabilistic.

Returns

True iff the sequence is probabilistic.

Implements Sequence.

Definition at line 241 of file sequence.h.

3.30.3.11 void SequenceFwd::set (const std::string & *seq*, bool *rev*) [virtual]

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.

Implements Sequence.

Definition at line 63 of file sequence.cpp.

3.30.3.12 void SequenceFwd::update_est (const size_t *index*, const size_t *nuc*, float *mass*) [virtual]

A member function that updates the posterior nucleotide distribution if probabilistic.

Parameters

index	the index of the position to update.
nuc	the nucleotide to increment the mass of.
mass	the amount to increment the nucleotide's mass by.

Implements Sequence.

Definition at line 104 of file sequence.cpp.

3.30.3.13 void SequenceFwd::update_exp (const size_t *index*, const size_t *nuc*, float *mass*) [virtual]

A member function that updates the expected frequency distribution if probabilistic.

Parameters

index	the index of the position to update.
nuc	the nucleotide to increment the mass of.

mass	the amount to increment the nucleotide's mass by.

Implements Sequence.

Definition at line 114 of file sequence.cpp.

3.30.3.14 void SequenceFwd::update_obs (const size_t *index*, const size_t *nuc*, float *mass*) [virtual]

A member function that updates the observed frequency distribution if probabilistic.

Parameters

index	the index of the position to update.
nuc	the nucleotide to increment the mass of.
mass	the amount to increment the nucleotide's mass by.

Implements Sequence.

Definition at line 109 of file sequence.cpp.

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

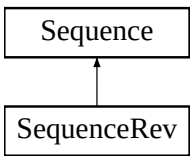
- src/sequence.h
- src/sequence.cpp

3.31 SequenceRev Class Reference

The SequenceRev class implements the Sequence abstract class for accessing the reverse sequence.

`#include <sequence.h>`

Inheritance diagram for SequenceRev:



Public Member Functions

- **SequenceRev** (SequenceFwd &seq)
- size_t length () const

  An accessor for the length of the encoded sequence.

- bool empty () const

  Accessor to determine if the sequence has 0 length.

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

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

- size_t operator[ ] (const size_t index) const

  An accessor for the encoded character at the given index.

- size_t get_ref (const size_t index) const

  An accessor for the encoded reference character at the given index.

- float get_obs (const size_t index, const size_t nuc) const

  An accessor for the observed nucleotide frequency (logged).

- float get_exp (const size_t index, const size_t nuc) const

  An accessor for the expected nucleotide frequency (logged).

- void update_est (const size_t index, const size_t nuc, float mass)

  A member function that updates the posterior nucleotide distribution if probabilistic.

- void update_obs (const size_t index, const size_t nuc, float mass)

  A member function that updates the observed frequency distribution if probabilistic.

- void update_exp (const size_t index, const size_t nuc, float mass)

  A member function that updates the expected frequency distribution if probabilistic.

- float get_prob (const size_t index, const size_t nuc) const

  An accessor for the posterior nucleotide distribution (logged).

- bool prob () const

  Accessor to determine if the sequence is probabilistic.

- void calc_p_vals (std::vector< double > &p_vals) const

  A member function that calculates p-values based on the observed and expected nucleotide frequences for the sequence.

3.31.1 Detailed Description

The SequenceRev class implements the Sequence abstract class for accessing the reverse sequence. Documentation is only provided for methods not documented in the abstract Sequence class and SequenceFwd class. This class acts by storing a pointer to a SequenceFwd object and reverse complementing

the input and output appropriately.

Author

Adam Roberts

Date

2012 Artistic License 2.0

Definition at line 258 of file sequence.h.

3.31.2 Member Function Documentation

3.31.2.1 void SequenceRev::calc_p_vals (std::vector< double > & *p_vals*) const [virtual]

A member function that calculates p-values based on the observed and expected nucleotide frequences for the sequence.

Experimental.

Parameters

p_vals	a reference to an empty vector of doubles to fill with p-values at each position.

Implements Sequence.

Definition at line 119 of file sequence.cpp.

3.31.2.2 bool SequenceRev::empty ( ) const [inline, virtual]

Accessor to determine if the sequence has 0 length.

Returns

True iff the sequence has 0 length.

Implements Sequence.

Definition at line 274 of file sequence.h.

3.31.2.3 float SequenceRev::get_exp (const size_t *index*, const size_t *nuc*) const [inline, virtual]

An accessor for the expected nucleotide frequency (logged).

Parameters

index	the index of the position to access.
nuc	the nucleotide to return the frequency of.

Returns

The logged expected frequency of the given nucleotide a the given position.

Implements Sequence.

Definition at line 288 of file sequence.h.

3.31.2.4 float SequenceRev::get_obs (const size_t *index*, const size_t *nuc*) const [inline, virtual]

An accessor for the observed nucleotide frequency (logged).

Parameters

index	the index of the position to access.
nuc	the nucleotide to return the frequency of.

Returns

The logged observed frequency of the given nucleotide a the given position.

Implements Sequence.

Definition at line 286 of file sequence.h.

3.31.2.5 float SequenceRev::get_prob (const size_t *index*, const size_t *nuc*) const [inline, virtual]

An accessor for the posterior nucleotide distribution (logged).

Parameters

index	the index of the position to access.
nuc	the nucleotide to return the probability of.

Returns

The logged posterior probability of the given nucleotide at the given position.

Implements Sequence.

Definition at line 296 of file sequence.h.

3.31.2.6 size_t SequenceRev::get_ref (const size_t *index*) const [inline, virtual]

An accessor for the encoded reference character at the given index.

May differ from the operator[] if the sequence is probabilistic.

Parameters

index	the index of the encoded reference character to return (assumed to be < _len)

Returns

The encoded reference character at the given index.

Implements Sequence.

Definition at line 284 of file sequence.h.

3.31.2.7 size_t SequenceRev::length ( ) const [inline, virtual]

An accessor for the length of the encoded sequence.

Returns

The length of the encoded sequence.

Implements Sequence.

Definition at line 268 of file sequence.h.

3.31.2.8 size_t SequenceRev::operator[] (const size_t *index*) const [inline, virtual]

An accessor for the encoded character at the given index.

If the sequence is probabilistic, returns the mode. Otherwise, returns the reference.

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 282 of file sequence.h.

3.31.2.9 bool SequenceRev::prob ( ) const [inline, virtual]

Accessor to determine if the sequence is probabilistic.

Returns

True iff the sequence is probabilistic.

Implements Sequence.

Definition at line 298 of file sequence.h.

3.31.2.10 void SequenceRev::set (const std::string & *seq*, bool *rev*) [inline, virtual]

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.

Implements Sequence.

Definition at line 281 of file sequence.h.

3.31.2.11 void SequenceRev::update_est (const size_t *index*, const size_t *nuc*, float *mass*) [inline, virtual]

A member function that updates the posterior nucleotide distribution if probabilistic.

Parameters

index	the index of the position to update.
nuc	the nucleotide to increment the mass of.
mass	the amount to increment the nucleotide's mass by.

Implements Sequence.

Definition at line 290 of file sequence.h.

3.31.2.12 void SequenceRev::update_exp (const size_t *index*, const size_t *nuc*, float *mass*) [inline, virtual]

A member function that updates the expected frequency distribution if probabilistic.

Parameters

index	the index of the position to update.
nuc	the nucleotide to increment the mass of.
mass	the amount to increment the nucleotide's mass by.

Implements Sequence.

Definition at line 294 of file sequence.h.

3.31.2.13 void SequenceRev::update_obs (const size_t *index*, const size_t *nuc*, float *mass*) [inline, virtual]

A member function that updates the observed frequency distribution if probabilistic.

Parameters

index	the index of the position to update.
nuc	the nucleotide to increment the mass of.
mass	the amount to increment the nucleotide's mass by.

Implements Sequence.

Definition at line 292 of file sequence.h.

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

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

3.32 SeqWeightTable Class Reference

The SeqWeightTable class keeps track of sequence-specific bias parameters.

`#include <biascorrection.h>`

Public Member Functions

- SeqWeightTable (size_t window_size, size_t order, double alpha)

  SeqWeightTable Constructor.

- SeqWeightTable (size_t window_size, size_t order, std::string param_file_name, std::string identifier)

  A second constructor that loads the distribution from a parameter file.

- void copy_observed (const SeqWeightTable &other)

  A member function that overwrites the "observed" parameters with those from another SeqWeightTable.

- void copy_expected (const SeqWeightTable &other)

  A member function that overwrites the "expected" parameters with those from another SeqWeightTable.

- void 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.

- void normalize_expected ()

  A member function that normalizes the expected counts and fills in the lower-ordered marginals.

- void 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 (logged) mass.

- double get_weight (const Sequence &seq, size_t i) const

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

- void append_output (std::ofstream &outfile) const

  A member function that appends the marginal and conditional probabilities for the foreground and background Markov models to the given file, formatted in tables for easy readability.

3.32.1 Detailed Description

The SeqWeightTable class keeps track of sequence-specific bias parameters. 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 are stored in log space.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 33 of file biascorrection.h.

3.32.2 Constructor & Destructor Documentation

3.32.2.1 SeqWeightTable::SeqWeightTable (size_t *window_size*, size_t *order*, double *alpha*)

SeqWeightTable Constructor.

Parameters

window_size	an unsigned integer specifying the size of the bias window surrounding fragment ends.
order	a size_t specifying the order to use for the Markov chains modelling the sequence.
alpha	a double specifying the strength of the uniform prior (logged pseudo-counts for each parameter).

Definition at line 31 of file biascorrection.cpp.

3.32.2.2 SeqWeightTable::SeqWeightTable (size_t *window_size*, size_t *order*, std::string *param_file_name*, std::string *identifier*)

A second constructor that loads the distribution from a parameter file.

Note that the values should not be modified after using this constructor.

Parameters

window_size	an unsigned integer specifying the size of the bias window surrounding fragment ends. Must match file.
order	a size_t specifying the order to use for the Markov chains modelling the sequence. Must match file.
param_file_name	a string specifying the path to the parameter file.
identifier	a string specifying the header for these parameters in the file.

3.32.3 Member Function Documentation

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

A member function that appends the marginal and conditional probabilities for the foreground and background Markov models to the given file, formatted in tables for easy readability.

Parameters

outfile	the file to append to.

3.32.3.2 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 95 of file biascorrection.cpp.

3.32.3.3 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 91 of file biascorrection.cpp.

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

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

This is the ratio of the observed and expected weights given by the two Markov models.

Parameters

seq	the target sequence.
i	the central point of the bias window, ie the fragment end.

Returns

The bias weight for the window.

Definition at line 114 of file biascorrection.cpp.

3.32.3.5 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 to increment by in the parameter table.
fl_cdf	the fragment length CDF.

Definition at line 99 of file biascorrection.cpp.

3.32.3.6 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 (logged) mass.

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, ie where the fragment starts/ends.
mass	the amount to increment by (logged)

Definition at line 108 of file biascorrection.cpp.

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

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

3.33 Target Class Reference

The Target class is used to store objects for the targets being mapped to.

`#include <targets.h>`

Public Member Functions

- Target (TargID id, const std::string &name, const std::string &seq, bool prob_-seq, double alpha, const Librarian *libs, const BiasBoss *known_bias_boss, const LengthDistribution *known_fld)

  Target Constructor.

- void lock () const

  A member function that locks the target mutex to provide thread safety.

- void unlock () const

  A member function that unlocks the target mutex.

- const std::string & name () const

  An accessor for the target name.

- TargID id () const

  An accessor for the target id.

- const Sequence & seq (bool rev=false) const

  An accessor for the the target's Sequence (const).

- Sequence & seq (bool rev)

  An accessor for the the target's Sequence (non-const).

- void haplotype (boost::shared_ptr< HaplotypeHandler > hh)

  Mutator for the HaplotypeHandler of the target.

- void alpha (double alpha)

  Mutator for the alpha (prior count) parameter of the target.

- size_t length () const

  An accessor for the length of the target sequence.

- double rho () const

  An accessor for he current estimated relative abundance of the target.

- double mass (bool with_pseudo=true) const

  An accessor for the current (logged) probabilistically assigned fragment mass.

- double mass_var () const

  An accessor for the total (logged) variance on mass.

- double var_sum () const

  An accessor for the (logged) weighted sum of the variance on assignments.

- double tot_ambig_mass () const

  An accessor for the the (logged) total mass of ambiguous fragments mapping to the target.

- void round_reset ()

  A member function that prepares the target object for the next round of batch EM.

- size_t tot_counts () const

  An accessor for the current count of fragments mapped to this target either uniquely or ambiguously.

- size_t uniq_counts () const

  An accessor for the the current count of fragments uniquely mapped to this target.

- Bundle * bundle () const

  An accessor for the pointer to the Bundle this Target is a member of.

- void bundle (Bundle *b)

  A mutator to set the Bundle this Target is a member of.

- void add_hit (const FragHit &h, double v, double mass)

  A member function that increases the expected fragment counts and variance based on the assignment parameters of the given FagHit.

- void incr_counts (bool uniq, size_t incr_amt=1)

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

- double sample_likelihood (bool with_pseudo, const std::vector< const Target * > *neighbors=NULL) const

  A member function that returns (a value proportional to) the probability of randomly sampling a fragment from the target.

- double align_likelihood (const FragHit &frag) const

A member function that returns (a value proportional to) the log likelihood the given fragment has the given alignment.

- double est_effective_length (const LengthDistribution *fld=NULL, bool with_bias=true) const

  A member function that calculates and returns the estimated effective length of the target (logged) using the average bias.

- double cached_effective_length (bool with_bias=true) const

  An accessor for the most recently estimated effective length (logged) as calculated by the bias updater thread.

- void update_target_bias_buffer (const BiasBoss *bias_table=NULL, const LengthDistribution *fld=NULL)

  A member function that causes the target bias to be re-calculated by the _bias_table based on curent parameters.

- void swap_bias_parameters ()

  Swaps in the buffered bias parameters for atomic updating.

- bool solvable () const

  An accessor for the _solvable flag.

- void solvable (bool s)

  A mutator that sets the _solvable flag.

Friends

- class HaplotypeHandler
- class TargetTable

3.33.1 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 as well as parameters for variance. To help with updating these values, it computes 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 90 of file targets.h.

3.33.2 Constructor & Destructor Documentation

3.33.2.1 Target::Target (TargID *id*, const std::string & *name*, const std::string & *seq*, bool *prob_seq*, double *alpha*, const Librarian * *libs*, const BiasBoss * *known_bias_boss*, const LengthDistribution * *known_fld*)

Target Constructor.

Parameters

id	a unique TargID identifier.
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).
libs	a pointer to the struct containing pointers to the global parameter tables (bias_table, mismatch_table, fld).
known_bias_boss	a pointer to bias parameters provided as input, NULL if none given.
known_fld	a pointer to a fragment length distribution provided as input, NULL if none given.

Definition at line 26 of file targets.cpp.

3.33.3 Member Function Documentation

3.33.3.1 void Target::add_hit (const FragHit & *h*, double *v*, double *mass*)

A member function that increases the expected fragment counts and variance based on the assignment parameters of the given FagHit.

Parameters

h	the FragHit that is being added.
v	a double for the (logged) approximate variance (uncertainty) on the probability p.
mass	a double specifying the (logged) mass of the fragment being mapped.

Definition at line 52 of file targets.cpp.

3.33.3.2 double Target::align_likelihood (const FragHit & *frag*) const

A member function that returns (a value proportional to) the log likelihood the given fragment has the given alignment.

Parameters

frag	a FragHit alignment to return the likelihood of.

Definition at line 129 of file targets.cpp.

3.33.3.3 void Target::alpha (double *alpha*) [inline]

Mutator for the alpha (prior count) parameter of the target.

Parameters

hh	non-logged value to set alpha to.

Definition at line 272 of file targets.h.

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

A mutator to 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 336 of file targets.h.

3.33.3.5 Bundle* Target::bundle ( ) const [inline]

An accessor for the pointer to the Bundle this Target is a member of.

Returns

A pointer to the Bundle this target is a member of.

Definition at line 331 of file targets.h.

3.33.3.6 double Target::cached_effective_length (bool *with_bias* = true) const

An accessor for the most recently estimated effective length (logged) as calculated by the bias updater thread.

Parameters

with_bias	a boolean specifying whether or not the average bias should be included in the return value.

Returns

The cached effective length of the target.

Definition at line 186 of file targets.cpp.

3.33.3.7 double Target::est_effective_length (const LengthDistribution * *fld* = NULL, bool *with_bias* = true) const

A member function that calculates and returns the estimated effective length of the target (logged) using the average bias.

Parameters

fld	an optional pointer to a different LengthDistribution than the global one, for thread-safety.
with_bias	a boolean specifying whether or not the average bias should be included in the return value.

Returns

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

Definition at line 162 of file targets.cpp.

3.33.3.8 void Target::haplotype (boost::shared_ptr< HaplotypeHandler > *hh*) [inline]

Mutator for the HaplotypeHandler of the target.

Parameters

hh	a shared pointer to the HaplotypeHandler.

Definition at line 265 of file targets.h.

3.33.3.9 TargID Target::id ( ) const [inline]

An accessor for the target id.

Returns

The target ID.

Definition at line 238 of file targets.h.

3.33.3.10 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 354 of file targets.h.

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

An accessor for the length of the target sequence.

Returns

The target sequence length.

Definition at line 277 of file targets.h.

3.33.3.12 void Target::lock ( ) const [inline]

A member function that locks the target mutex to provide thread safety.

The lock should be held by any thread that calls a method of the Target.

Definition at line 224 of file targets.h.

3.33.3.13 double Target::mass (bool *with_pseudo* = true) const

An accessor for the current (logged) probabilistically assigned fragment mass.

Parameters

with_pseudo	a boolean specifying whether pseudo-counts should be included in returned mass.

Returns

The logged mass.

Definition at line 99 of file targets.cpp.

3.33.3.14 double Target::mass_var ( ) const

An accessor for the total (logged) variance on mass.

Includes variance due to the initial pseudo-counts.

Returns

The total (logged) variance on mass.

Definition at line 106 of file targets.cpp.

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

An accessor for the target name.

Returns

string containing target name.

Definition at line 233 of file targets.h.

3.33.3.16 double Target::rho ( ) const

An accessor for he current estimated relative abundance of the target.

The value is logged and includes pseudo-counts.

Returns

The current estimated rho.

Definition at line 90 of file targets.cpp.

3.33.3.17 double Target::sample_likelihood (bool *with_pseudo*, const std::vector< const Target * > * *neighbors* = `NULL`) const

A member function that returns (a value proportional to) the probability of randomly sampling a fragment from the target.

Parameters

with_pseudo	a bool specifying whether or not pseudo-counts should be included in the calculation.
neighbors	a vector of Target pointers for neighbors to include in the binned likelihood.

Returns

A value proportional to the log likelihood the given fragment originated from this target.

Definition at line 110 of file targets.cpp.

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

An accessor for the the target's Sequence (const).

Parameters

rev	a bool specifying whether to return the reverse complement.

Returns

Const reference to the target's Sequence object.

Definition at line 244 of file targets.h.

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

An accessor for the the target's Sequence (non-const).

Parameters

rev	a bool specifying whether to return the reverse complement.

Returns

Non-const reference to the target's Sequence object.

Definition at line 255 of file targets.h.

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

A mutator that sets the _solvable flag.

Parameters

a	boolean specifying whether or not the target has a unique solution for its abundance estimate.

Definition at line 428 of file targets.h.

3.33.3.21 bool Target::solvable ( ) const [inline]

An accessor for the _solvable flag.

Returns

a boolean specifying whether or not the target has a unique solution for its abundance estimate.

Definition at line 422 of file targets.h.

3.33.3.22 void Target::swap_bias_parameters ( )

Swaps in the buffered bias parameters for atomic updating.

The target mutex should be held by the caller.

Definition at line 203 of file targets.cpp.

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

An accessor for the 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 309 of file targets.h.

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

An accessor for the current count of fragments mapped to this target either uniquely or ambiguously.

Returns

The total fragment count.

Definition at line 320 of file targets.h.

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

An accessor for the the current count of fragments uniquely mapped to this target.

Returns

The unique fragment count.

Definition at line 326 of file targets.h.

3.33.3.26 void Target::update_target_bias_buffer (const BiasBoss * *bias_table* = NULL, const LengthDistribution * *fld* = NULL)

A member function that causes the target bias to be re-calculated by the _bias_table based on curent parameters.

The results are buffered until swap_bias_parameters is called to allow for atomic updating.

Parameters

bias_table	a pointer to a BiasBoss to use as parameters. Bias not updated if NULL.
fld	an optional pointer to a different LengthDistribution than the global one, for thread-safety.

Definition at line 193 of file targets.cpp.

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

An accessor for the (logged) weighted sum of the variance on assignments.

Returns

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

Definition at line 302 of file targets.h.

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

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

3.34 TargetTable Class Reference

The TargetTable class is used to keep track of the Target objects for a run.

`#include <targets.h>`

Public Member Functions

- TargetTable (std::string targ_fasta_file, std::string haplotype_file, bool prob_-seqs, bool known_aux_params, double alpha, const AlphaMap *alpha_map, const Librarian *libs)

  TargetTable Constructor.

- ~TargetTable ()

  TargetTable Destructor.

- Target * get_targ (TargID id)

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

- void round_reset ()

  A member function that readies all Target objects in the table for the next round of batch EM.

- size_t size () const

  An accessor for the number of targets in the table.

- double total_fpb () const

  An accessor for the (logged) total mass per base, including pseudo-counts.

- void update_total_fpb (double incr_amt)

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

- void update_covar (TargID targ1, TargID targ2, double covar)

  A member function that increases the (logged) covariance between two targets by the specified amount.

- double get_covar (TargID targ1, TargID targ2)

  An accessor for the covariance between two targets.

- size_t covar_size () const

  An accessor for number of pairs of targets with non-zero covariance.

- Bundle * merge_bundles (Bundle *b1, Bundle *b2)

  A member function that merges the given Bundles.

- size_t num_bundles () const

  An accessor for the number of bundles in the partition.

- void masses_to_counts ()

  Renormalized masses to be counts and projects when necessary.

- void output_results (std::string output_dir, size_t tot_counts, bool output_varcov=false, bool output_rdds=false)

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

- void asynch_bias_update (boost::mutex *mutex)

  A member function to be run asynchronously that continuously updates the background bias values, target bias values, and target effective lengths.

- void **enable_bundle_threadsafety** ()
- void **disable_bundle_threadsafety** ()
- void collapse_bundles ()

  Collapses the merge trees in the BundleTable.

3.34.1 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 stores them in a map keyed by their string id.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 520 of file targets.h.

3.34.2 Constructor & Destructor Documentation

3.34.2.1 TargetTable::TargetTable (std::string *targ_fasta_file*, std::string *haplotype_file*, bool *prob_seqs*, bool *known_aux_params*, 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.
haplotype_file	a string storing the path to the haplotype file containing comma-separated target pairs to be considered alternative haplotypes (optional).
prob_seqs	a bool that specifies if the sequence is to be treated probablistically, for RDD detection.
known_aux_params	a bool that is true iff the auxiliary parameters (fld, bias) are provided and need not be learned.
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.
libs	a pointer to the struct containing pointers to the global parameter tables (bias_table, mismatch_table, fld).

Definition at line 292 of file targets.cpp.

3.34.2.2 TargetTable::~TargetTable ( )

TargetTable Destructor.

Deletes all of the target objects in the table.

Definition at line 424 of file targets.cpp.

3.34.3 Member Function Documentation

3.34.3.1 void TargetTable::asynch_bias_update (boost::mutex * *mutex*)

A member function to be run asynchronously that continuously updates the background bias values, target bias values, and target effective lengths.

Parameters

mutex	a pointer to the mutex to be used to protect the global fld and bias tables during updates.

Definition at line 843 of file targets.cpp.

3.34.3.2 size_t TargetTable::covar_size ( ) const [inline]

An accessor for number of pairs of targets with non-zero covariance.

Returns

The number of target pairs with non-zero covariance.

Definition at line 652 of file targets.h.

3.34.3.3 double TargetTable::get_covar (TargID *targ1*, TargID *targ2*) [inline]

An accessor for 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 645 of file targets.h.

3.34.3.4 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

A pointer to the target with the given id.

Definition at line 462 of file targets.cpp.

3.34.3.5 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 466 of file targets.cpp.

3.34.3.6 size_t TargetTable::num_bundles ( ) const [inline]

An accessor for the number of bundles in the partition.

Returns

The number of bundles in the partition.

Definition at line 664 of file targets.h.

3.34.3.7 void TargetTable::output_results (std::string *output_dir*, size_t *tot_counts*, bool *output_varcov* = false, bool *output_rdds* = false)

A member function that outputs the final expression data in a file called 'results.xprs', (optionally) the variance-covariance matrix in 'varcov.xprs', and (optionally) the RDD p-values 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_rdds	boolean specifying whether to also output the RDD p-values.

Definition at line 588 of file targets.cpp.

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

An accessor for the number of targets in the table.

Returns

The number of targets in the table.

Definition at line 614 of file targets.h.

3.34.3.9 double TargetTable::total_fpb ( ) const

An accessor for the (logged) total mass per base, including pseudo-counts.

Returns

The (logged) total mass per base, including pseudo-counts.

Definition at line 833 of file targets.cpp.

3.34.3.10 void TargetTable::update_covar (TargID *targ1*, TargID *targ2*, double *covar*) [inline]

A member function that increases the (logged) covariance between two targets by the specified amount.

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 635 of file targets.h.

3.34.3.11 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 838 of file targets.cpp.

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

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

3.35 ThreadSafeFragQueue Class Reference

The ThreadSafeFragQueue is a threadsafe queue of Fragment pointers.

`#include <threadsafety.h>`

Public Member Functions

- ThreadSafeFragQueue (size_t max_size)

  ThreadSafeFragQueue Constructor.

- Fragment * pop (bool block=true)

  A member function that pops the next Fragment pointer off the queue.

- void push (Fragment *frag)

  A member function that pushes the given Fragment pointer onto the queue.

- bool is_empty (bool block=false)

  A member function that returns true iff the queue is empty.

3.35.1 Detailed Description

The ThreadSafeFragQueue is a threadsafe queue of Fragment pointers.

Author

Adam Roberts

Date

2012 Artistic License 2.0

Definition at line 23 of file threadsafety.h.

3.35.2 Constructor & Destructor Documentation

3.35.2.1 ThreadSafeFragQueue::ThreadSafeFragQueue (size_t *max_size*)

ThreadSafeFragQueue Constructor.

Parameters

max_size	a size_t representing the number of Fragments allowed in

the queue before blocking on a push.

Definition at line 12 of file threadsafety.cpp.

3.35.3 Member Function Documentation

3.35.3.1 bool ThreadSafeFragQueue::is_empty (bool *block* = `false`)

A member function that returns true iff the queue is empty.

If block is true, the function blocks until the queue is empty and returns true.

Parameters

block	a bool specifying whether or not the function should block until the queue is empty.

Returns

True iff the queue is empty.

Definition at line 41 of file threadsafety.cpp.

3.35.3.2 Fragment * ThreadSafeFragQueue::pop (bool *block* = `true`)

A member function that pops the next Fragment pointer off the queue.

If the queue is empty, returns NULL if block is false, otherwise blocks until one is available.

Parameters

block	a bool specifying whether or not the function should block if the queue is empty.

Returns

The next Fragment pointer on the queue or NULL if the queue is empty and block is false.

Definition at line 16 of file threadsafety.cpp.

3.35.3.3 void ThreadSafeFragQueue::push (Fragment * *frag*)

A member function that pushes the given Fragment pointer onto the queue.

Blocks if the queue is full.

Parameters

frag	the Fragment pointer to push onto the queue.

Definition at line 31 of file threadsafety.cpp.

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

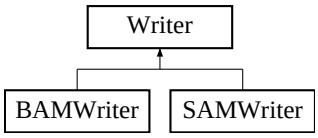
- src/threadsafety.h
- src/threadsafety.cpp

3.36 Writer Class Reference

The Writer class is an abstract class for implementing a SAMWriter or BAMWriter.

`#include <mapparser.h>`

Inheritance diagram for Writer:



Public Member Functions

- virtual ~Writer ()

  Dummy destructor.

- virtual void write_fragment (Fragment &f)=0

  A member function that writes all mappings of the fragment to the ouptut file along with their posterior probabilities in the "XP" field.

Protected Attributes

- bool _sample

  A private bool that specifies if a single alignment should be sampled (true) or all output with their respective posterior probabilities (false).

3.36.1 Detailed Description

The Writer class is an abstract class for implementing a SAMWriter or BAMWriter. It writes Fragment objects back to file (in SAM/BAM format) with per-mapping probabilistic assignments, or by sampling a single mapping based on assignment probabilities.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 99 of file mapparser.h.

3.36.2 Member Function Documentation

3.36.2.1 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 posterior 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