Extract polymorphism/divergence

Generated by Doxygen 1.8.20

Chapter 1

Main Page

This set of software extracts polymorphism/divergence data from various files. These data will be used to do Mac↩ Donald-Kreitman type tests. This is for a specific project and data set. Generalization is possible, but may require modifying the code.

1.1 Installation

Clone this repository, change directory to `polyDivExtract` and type

```
make
make install clean
```

This assumes that you are using `g++` as the C++ compiler. To use a different compiler, specify it on the `make` command line, e.g.

```
make CXX=c++
```

1.1.1 Dependencies

No dependencies other than a C++ compiler that understands the C++11 standard.

1.2 Usage

The `divSites` program takes nucleotide positions or ranges and returns a file with sites that have diverged between two species, inferred from the provided AXT between-species alignment file. Run it with

```
divSites -q file_with_positions -a AXT_alignment_file -o output_file
```

The query files should have at least two fields (chromosome ID and position). Chromosome IDs must be chrX, chr2L, chr2R, chr3L, or chr3R (this is for a *Drosophila* data set). It is OK to omit "chr" from the chromosome arm names. Chromosomes must be listed in the same order in all files. The AXT file should have the same chromosome names as the query file. If there are exactly two fields, it is assumed that the file provides individual site positions ("positions file"). If there are more than two fields, it is assumed that the query file contains ranges of positions, with the first field indicating the chromosome arm, the second the start of the range, and the third the end. If there are more that three fields, the rest are ignored. Commented (starting with "#") and empty lines are ignored. The number of fields is checked on the first uncommented non-empty line. This line can be a header (defined as having non-numeric values in the position or start and/or end fields). It must have two fields for a positions file or no fewer than three fields for a ranges file. If the query file contains positions, the output file has the chromosome ID, position, focal species nucleotide, alternative (diverged) nucleotide, whether the alternative is on the same chromosome (1

if yes), and whether both nucleotides are good quality (1 if yes). The total number of good quality nucleotides per chromosome is listed as meta-data (commented out with #) at the start if the file. Of the query file has ranges, the output is similar but lists the "peak ID" (corresponding to each range) and number of good quality nucleotides in the range before the fields listed above, and no meta-data.

The `polySites` program extracts polymorphic sites. Run it with
```
polySites -q query_file -a AXT_alignment_file -v VCF_file -o output_file
```

Query files are the same as for `divSites`, as are the AXT files (the latter are used to call outgroup states). The VCF file contains polymorphism information. Chromosomes must be labeled the same as in the AXT and query files (with or without "chr" in front). The output file for position queries has the chromosome ID, position, reference nucleotide, alternative nucleotide, ancestral state (`r` if reference, `a` if alternative), derived allele count, maximum likelihood derived allele count, derived allele frequency, maximum likelihood derived allele frequency, number of missing genotypes, whether the outgroup site is on the same chromosome (1 if yes), whether the outgroup nucleotide is good quality (1 if yes), and the site quality score. The output is similar for a range query file, but includes "peak ID" (i.e., range ID).

The `fastaSort` program sorts FASTA files that have *loc=* fields in their headers by start nucleotide position. If there are records with the same start position, only the longest one is kept. Run with
```
fastaSort -i input_FASTA -o output_FASTA
```

Having sorted a FASTA file with coding sequences, run
```
getFFsites -i input_sorted_FASTA -l log_file_name -o output_file
```

This extracts four-fold silent sites from each CDS, discarding regions of CDS overlap. The output lists the chromosome, FBgn number of the CDS, and chromosome position of the site. The log file contains debugging information, flags overlapping CDS, and highlights potentially problematic records.

The software assumes *Drosophila* chromosomes, labeled as chrX, chr2L, etc.

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 BayesicSpace::FFextract Class Reference

Four-fold synonymous site extraction.

`#include <ffExtract.hpp>`

Public Member Functions

- FFextract ()
  Default constructor.
- FFextract (const string &fastaName, const string &logName)
  Constructor.
- ~FFextract ()
  Destructor.
- FFextract (const FFextract &in)=delete
  Copy constructor (deleted)
- FFextract (FFextract &&in)
  Move constructor.
- void extractFFsites (vector< string > &positionList)
  Extract four-fold sites from the current record.

4.1.1 Detailed Description

Four-fold synonymous site extraction.

The class reads a FASTA file with coding sequences and extracts four-fold synonymous sites. The algorithm assumes that the records are sorted by chromosome position of the start site. This can be achieved by running the enclosed `fastaSort` program. It is also assumed that there are no CDS that completely within other genes. The `fastaSort` program eliminates such cases. We also assume that the sequence portions of FASTA records are all on one line. This is how `fastaSort` outputs them. The chromosome names are for Drosophila (2L, 2R, 3L, 3R, 4, X). The names may be preceded by the Scf_ prefix, which is used in the D. simulans genome. Output chromosome names are the Drosophila set preceded by `chr`.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 FFextract() [1/2]

```
FFextract::FFextract (
            const string & fastaName,
            const string & logName )
```

Constructor.

Parameters

in	*fastaName*	name of the FASTA file
in	*logName*	name of the log file

4.1.2.2 FFextract() [2/2]

```
BayesicSpace::FFextract::FFextract (
            FFextract && in )  [inline]
```

Move constructor.

Parameters

in	*in*	the object to be moved

4.1.3 Member Function Documentation

4.1.3.1 extractFFsites()

```
void FFextract::extractFFsites (
            vector< string > & positionList )
```

Extract four-fold sites from the current record.

The vector of positions contains tab-delimited fields: chromosome, FBgn number, and position. The contents of the vector are replaced.

Parameters

out	*positionList*	vector of four-fold site positions.

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

- ffExtract.hpp
- ffExtract.cpp

4.2 BayesicSpace::ParseAXT Class Reference

.axt alignment parsing class

`#include <parseAXT.hpp>`

Public Member Functions

- ParseAXT ()
  Default constructor.
- ParseAXT (const string &fileName)
  File name constructor.
- ~ParseAXT ()
  Destructor.
- ParseAXT (const ParseAXT &in)=delete
  Copy constructor.
- ParseAXT (ParseAXT &&in)
  Move constructor.
- ParseAXT & operator= (const ParseAXT &in)=delete
  Copy assignment.
- ParseAXT & operator= (ParseAXT &&in)
  Move assignment.
- string getMetaData ()
  Record meta data.
- string getPrimarySeq ()
  Get primary sequence.
- string getAlignedSeq ()
  Get aligned sequence.
- void getDivergedSites (const string &chromName, const uint64_t &start, const uint64_t &end, vector< string > &sites, uint64_t &length)
  Get list of divergent sites from a range.
- void getDivergedSites (const vector< string > &chromNames, const vector< uint64_t > &positions, vector< string > &sites, unordered_map< string, uint64_t > &lengths)
  Get list of divergent sites from a vector of positions.
- void getOutgroupState (const string &chromName, const uint64_t &position, string &site)
  Get the outgroup state for a position.

4.2.1 Detailed Description

.axt alignment parsing class

Exatracts features from an .axt alignement file.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 ParseAXT()

```
ParseAXT::ParseAXT (
            const string & fileName )
```

File name constructor.

Initializes the file stream and loads first AXT record.

Parameters

in	*fileName*	file name

4.2.3 Member Function Documentation

4.2.3.1 getAlignedSeq()

```
string BayesicSpace::ParseAXT::getAlignedSeq ( )  [inline]
```

Get aligned sequence.

Returns

string with the aligned sequence

4.2.3.2 getDivergedSites() [1/2]

```
void ParseAXT::getDivergedSites (
            const string & chromName,
            const uint64_t & start,
            const uint64_t & end,
            vector< string > & sites,
            uint64_t & length )
```

Get list of divergent sites from a range.

Get a list of divergent sites from a range of positions on a chromosome. Sites that are not covered or align to gaps are not counted in computing the overall length. The vector of sites is appended by the function, so any existing information will be preserved. The site description is in a tab-delimited string with the following fields:

- chromosome name
- position
- primary nucleotide
- aligned nucleotide
- whether the aligned nucleotide is on the same chromosome
- whether both nucleotides are in upper case (indicating high quality base calls)

NOTE: The range must be confined to a single chromosome.

Parameters

in	*chromName*	chromosome name
in	*start*	start position of the target range
in	*end*	end position of the target range
out	*sites*	vector of divergent site information (appended after execution)
out	*length*	length not counting sites that are missing or align to gaps

4.2.3.3 getDivergedSites() [2/2]

```
void ParseAXT::getDivergedSites (
            const vector< string > & chromNames,
            const vector< uint64_t > & positions,
            vector< string > & sites,
            unordered_map< string, uint64_t > & lengths )
```

Get list of divergent sites from a vector of positions.

Get a list of divergent sites from a vector of positions. The provided vector of cromosome names must be the same length as the vector of genome positions. The chromosome names must be arranged in contiguous blocks, with the same order as in the target .axt file. This is to speed up file traversal. Sites that are not covered or align to gaps are not counted in computing the overall length. The vector of sites is appended by the function, so any existing information will be preserved. The site description is in a tab-delimited string with the following fields:

- chromosome name
- position
- primary nucleotide
- aligned nucleotide
- whether the aligned nucleotide is on the same chromosome
- whether both nucleotides are in upper case (indicating high quality base calls)

Parameters

in	*chromNames*	vector of chromosome names
in	*positions*	vector of query site genome positions
out	*sites*	vector of divergent site information (appended after execution)
out	*lengths*	lengths, one per chromosome, not counting sites that are missing or align to gaps

4.2.3.4 getMetaData()

```
string ParseAXT::getMetaData ( )
```

Record meta data.

Returns a string with space-delimited metadata for the curent record:

- primary chromosome
- is the aligned chromosome the same (0/1)?
- primary start
- primary end
- aligned start
- aligned end

Returns

string with metadata

4.2.3.5 getOutgroupState()

```
void ParseAXT::getOutgroupState (
            const string & chromName,
            const uint64_t & position,
            string & site )
```

Get the outgroup state for a position.

The aligned genome is assumed to belong to the outgroup species. The site description is in a three-letter (no delimitation) string with the following fields:

- aligned (outgroup) nucleotide ("N" if not available)
- whether the aligned nucleotide is on the same chromosome
- whether the aligned nucleotide is in upper case (indicating high quality base calls)

Parameters

in	*chromName*	chromosome name
in	*position*	query site genome position
out	*site*	outgroup site information

4.2.3.6 getPrimarySeq()

```
string BayesicSpace::ParseAXT::getPrimarySeq ( )  [inline]
```

Get primary sequence.

Returns

string with the primary sequence

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

- parseAXT.hpp
- parseAXT.cpp

4.3 BayesicSpace::ParseVCF Class Reference

VCF file parsing class.

```
#include <parseVCF.hpp>
```

Public Member Functions

- ParseVCF ()
  Default constructor.
- ParseVCF (const string &vcfFileName, const string &axtFileName)
  Constructor with file names.
- ~ParseVCF ()
  Destructor.
- ParseVCF (const ParseVCF &in)=delete
  Copy constructor.
- ParseVCF (ParseVCF &&in)=delete
  Move constructor.
- ParseVCF & operator= (const ParseVCF &in)=delete
  Copy assignment.
- ParseAXT & operator= (ParseVCF &&in)=delete
  Move assignment.
- void getPolySites (const string &chromName, const uint64_t &start, const uint64_t &end, vector< string > &sites)
  Get list of polymorphic sites from a range.
- void getPolySites (const vector< string > &chromNames, const vector< uint64_t > &positions, vector< string > &sites)
  Get list of polymorphic sites from a vector of positions.

4.3.1 Detailed Description

VCF file parsing class.

Extracts information from a VCF file by position. Only SNPs are considered. The parsing is for the specific VCF files with fields defined in the dosage compensation project, may not be generally applicable.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 ParseVCF()

```
ParseVCF::ParseVCF (
            const string & vcfFileName,
            const string & axtFileName )
```

Constructor with file names.

Opens the VCF file and the corresponding .axt alignment file for ancestral state tracking.

Parameters

in	*vcfFileName*	name of the VCF file
in	*axtFileName*	name of the .axt file

4.3.3 Member Function Documentation

4.3.3.1 getPolySites() [1/2]

```
void ParseVCF::getPolySites (
            const string & chromName,
            const uint64_t & start,
            const uint64_t & end,
            vector< string > & sites )
```

Get list of polymorphic sites from a range.

Get a list of polymorphic sites from a range of positions on a chromosome. The vector of sites is appended by the function, so any existing information will be preserved. The site description is in a tab-delimited string with the following fields:

- chromosome name
- position
- reference nucleotide
- alternative nucleotide
- which nucleotide is ancestral ('r' for reference, 'a' alternative, 'u' unknown)
- derived allele count
- derived allele count (maximum likelihood)
- derived allele frequency

- derived allele frequencey (maximum likelihood)
- number of missing genotypes
- whether the outgroup nucleotide is on the same chromosome as the polymorphic site
- whether the outgroup nucleotide is good quality
- site quality score

NOTE: The range must be confined to a single chromosome.

Parameters

in	*chromName*	chromosome name
in	*start*	start position of the target range
in	*end*	end position of the target range
out	*sites*	vector of divergent site information (appended after execution)

4.3.3.2 getPolySites() [2/2]

```
void ParseVCF::getPolySites (
            const vector< string > & chromNames,
            const vector< uint64_t > & positions,
            vector< string > & sites )
```

Get list of polymorphic sites from a vector of positions.

Get a list of polymorphic sites from a vector of positions. The provided vector of cromosome names must be the same length as the vector of genome positions. The chromosome names must be arranged in contiguous blocks, with the same order as in the target VCF file. This is to speed up file traversal. The vector of sites is appended by the function, so any existing information will be preserved. The site description is in a tab-delimited string with the following fields:

- chromosome name
- position
- reference nucleotide
- alternative nucleotide
- which nucleotide is ancestral ('r' for reference, 'a' alternative, 'u' unknown)
- derived allele count
- derived allele count (maximum likelihood)
- derived allele frequency
- derived allele frequencey (maximum likelihood)
- number of missing genotypes
- whether the outgroup nucleotide is on the same chromosome as the polymorphic site
- whether the outgroup nucleotide is good quality
- site quality score

Parameters

in	*chromNames*	vector of chromosome names
in	*positions*	vector of query site genome positions
out	*sites*	vector of divergent site information (appended after execution)

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

- parseVCF.hpp
- parseVCF.cpp

Chapter 5

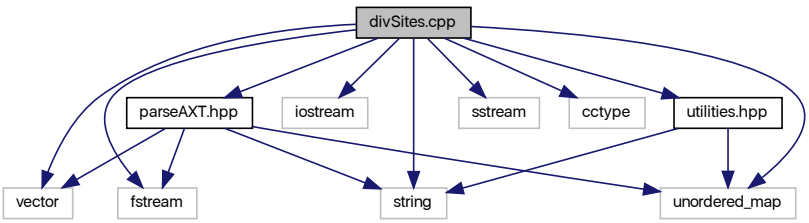
File Documentation

5.1 divSites.cpp File Reference

Extract diverged sites.

```
#include <string>
#include <vector>
#include <unordered_map>
#include <iostream>
#include <fstream>
#include <sstream>
#include <cctype>
#include "parseAXT.hpp"
#include "utilities.hpp"
```

Include dependency graph for divSites.cpp:



Functions

- int **main** (int argc, char ∗argv[ ])

5.1.1 Detailed Description

Extract diverged sites.

Author

Anthony J. Greenberg

Copyright

Version

0.1

Extracts divergent sites from MSL complex peak ranges and the four-fold silent site file list. The divergence is either to *D. simulans* or *D. yakuba*. The flags are:

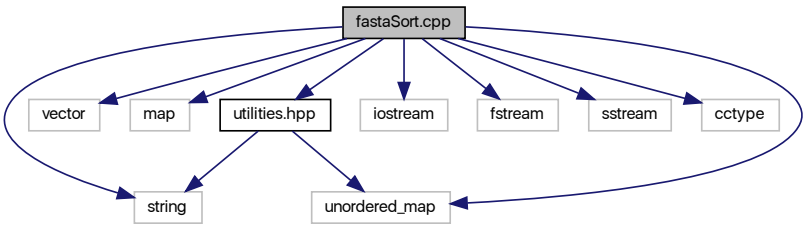
-q query file name (binding locations or four-fold sites) -a .axt file name -o output file name

5.2 fastaSort.cpp File Reference

Sort a FASTA file by chromosome and position.

```
#include <string>
#include <vector>
#include <map>
#include <unordered_map>
#include <iostream>
#include <fstream>
#include <sstream>
#include <cctype>
#include "utilities.hpp"
```

Include dependency graph for fastaSort.cpp:



Functions

- void getFBgnLastPos (const string &header, string &fbgn, uint64_t &lastPos)

  Extract FBgn number and last position.
- int **main** (int argc, char ∗argv[ ])

5.2.1 Detailed Description

Sort a FASTA file by chromosome and position.

Author

Anthony J. Greenberg

Copyright

Version

0.1

Sorts a FASTA file that has a *loc=* field in the header of each sequence by position (of the start nucleotide) within each chromosome. If records with the same position are found, the longest one is kept. If records have the same FBgn number, the longer one is kept. Any CDS that fall completely within another are eliminated. The flags are:

-i input file name -o output file name

5.2.2 Function Documentation

5.2.2.1 getFBgnLastPos()

```
void getFBgnLastPos (
            const string & header,
            string & fbgn,
            uint64_t & lastPos )
```

Extract FBgn number and last position.

Extracts the FBgn from the FASTA header.

Parameters

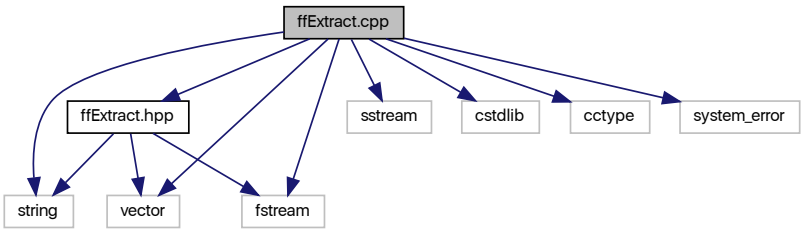
in	*header*	FASTA header
out	*fbgn*	FBgn number (excludes the *FBgn* string)
out	*lastPos*	last position in the CDS

5.3 ffExtract.cpp File Reference

Extract four-fold sites.

```
#include <string>
#include <vector>
#include <fstream>
#include <sstream>
#include <cstdlib>
#include <cctype>
#include <system_error>
#include "ffExtract.hpp"
```

Include dependency graph for ffExtract.cpp:



5.3.1 Detailed Description

Extract four-fold sites.

Author

Anthony J. Greenberg

Copyright

Version

0.1

Class implementation for extracting four-fold synonymous site positions from FASTA files.

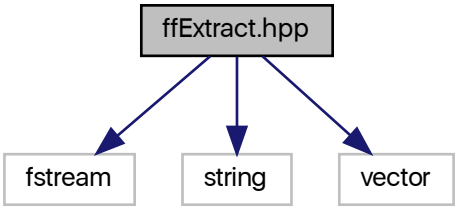
5.4 ffExtract.hpp File Reference

Extract four-fold sites.

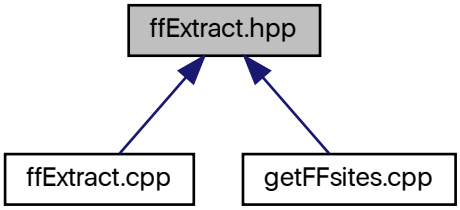
```
#include <fstream>
#include <string>
```

```
#include <vector>
```

Include dependency graph for ffExtract.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class BayesicSpace::FFextract

  Four-fold synonymous site extraction.

5.4.1 Detailed Description

Extract four-fold sites.

Author

Anthony J. Greenberg

Copyright

Copyright (c) 2019 Anthony J. Greenberg

Version

0.1

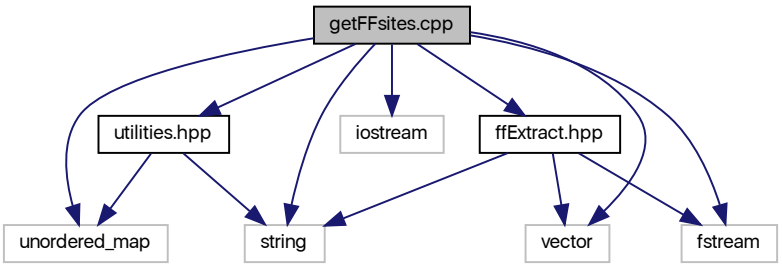
Class definitions and interface documentation for extracting four-fold synonymous site positions from FASTA files.

5.5 getFFsites.cpp File Reference

Extract four-fold synonymous sites from a CDS FASTA file.

```
#include <string>
#include <vector>
#include <unordered_map>
#include <iostream>
#include <fstream>
#include "utilities.hpp"
#include "ffExtract.hpp"
```

Include dependency graph for getFFsites.cpp:



Functions

- int **main** (int argc, char ∗argv[ ])

5.5.1 Detailed Description

Extract four-fold synonymous sites from a CDS FASTA file.

Author

Anthony J. Greenberg

Copyright

Copyright (c) 2019 Anthony J. Greenberg

Version

0.1

Takes a FASTA file with coding sequences (CDS), sorted by chromosome and position by `fastaSort`, and outputs a list of four-fold synonymous sites. Regions that are covered by overlapping CDS are discarded.

The flags are:

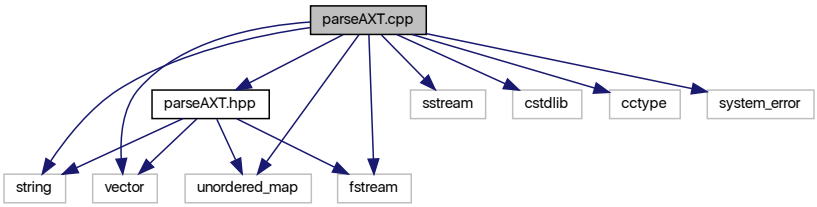
-i input file name -l log file name -o output file name

5.6 parseAXT.cpp File Reference

Parse .axt aligniment files.

```
#include <string>
#include <vector>
#include <unordered_map>
#include <fstream>
#include <sstream>
#include <cstdlib>
#include <cctype>
#include <system_error>
#include "parseAXT.hpp"
```

Include dependency graph for parseAXT.cpp:



5.6.1 Detailed Description

Parse .axt aligniment files.

Author

Anthony J. Greenberg

Copyright

Copyright (c) 2019 Anthony J. Greenberg

Version

0.1

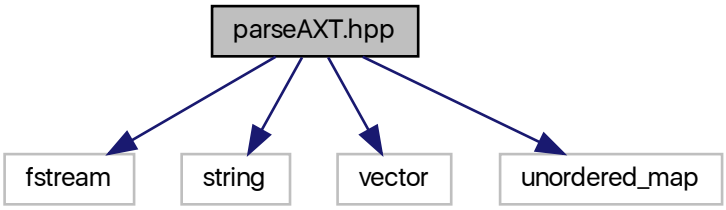
Class implementation for parsing of .axt alignement files to extract ancestral nucleotide and divergence information

5.7 parseAXT.hpp File Reference

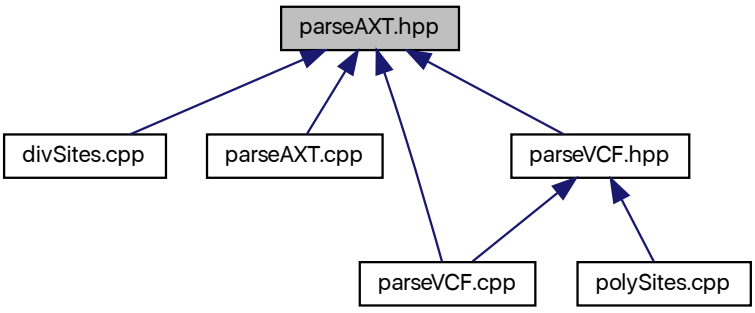
Parse .axt aligniment files.

```
#include <fstream>
#include <string>
#include <vector>
#include <unordered_map>
```

Include dependency graph for parseAXT.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class BayesicSpace::ParseAXT

  .axt alignment parsing class

5.7.1 Detailed Description

Parse .axt aligniment files.

Author

Anthony J. Greenberg

Copyright

Version

0.1

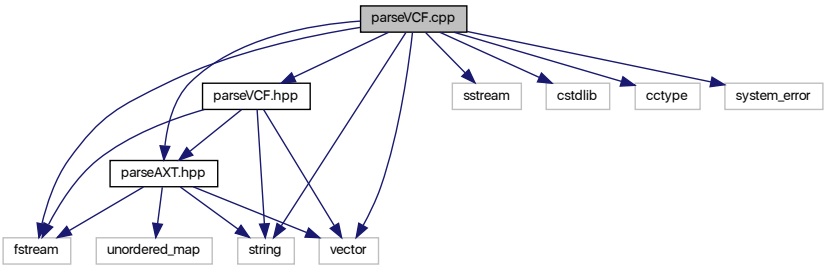
Class definitions and interface documentation for parsing of .axt alignement files to extract ancestral nucleotide and divergence information

5.8 parseVCF.cpp File Reference

Parse .vcf files.

```
#include <fstream>
#include <string>
#include <vector>
#include <sstream>
#include <cstdlib>
#include <cctype>
#include <system_error>
#include "parseVCF.hpp"
#include "parseAXT.hpp"
```

Include dependency graph for parseVCF.cpp:



5.8.1 Detailed Description

Parse .vcf files.

Author

Anthony J. Greenberg

Copyright

Copyright (c) 2019 Anthony J. Greenberg

Version

0.1

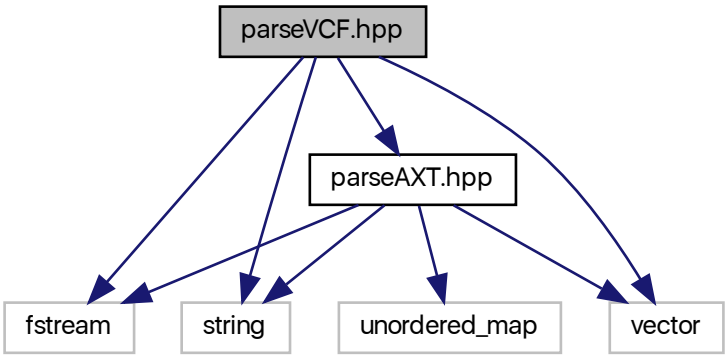
Class implementation for parsing of .vcf files to extract ancestral nucleotide and polymorphism information. This is specific to the dosage compensation evolution project.

5.9 parseVCF.hpp File Reference

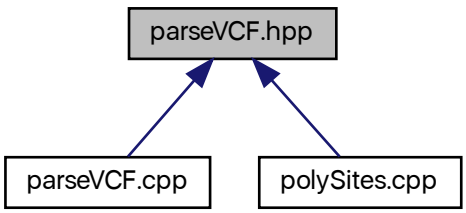
Parse .vcf files.

```
#include <fstream>
#include <string>
#include <vector>
#include "parseAXT.hpp"
```

Include dependency graph for parseVCF.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class BayesicSpace::ParseVCF
  VCF file parsing class.

5.9.1 Detailed Description

Parse .vcf files.

Author

Anthony J. Greenberg

Copyright

Version

0.1

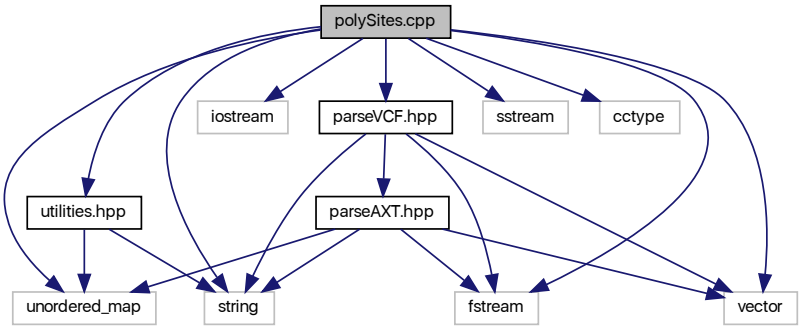
Class definitions and interface documentation for parsing of .vcf files to extract ancestral nucleotide and polymorphism information. This is specific to the dosage compensation evolution project.

5.10 polySites.cpp File Reference

Extract polymorphic sites.

```
#include <string>
#include <vector>
#include <unordered_map>
#include <iostream>
#include <fstream>
#include <sstream>
#include <cctype>
#include "parseVCF.hpp"
#include "utilities.hpp"
```

Include dependency graph for polySites.cpp:



Functions

- int **main** (int argc, char ∗argv[ ])

5.10.1 Detailed Description

Extract polymorphic sites.

Author

Anthony J. Greenberg

Copyright

Version

0.1

Extracts polymorphic sites from MSL complex peak ranges and the four-fold silent site file list. Outgroups for ancestral state determination is either *D. simulans* or *D. yakuba*. The flags are:

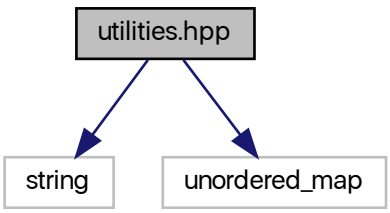
-q query file name (binding locations or four-fold sites) -a .axt file name (for the outgroup) -v VCF file name -o output file name

5.11 utilities.hpp File Reference

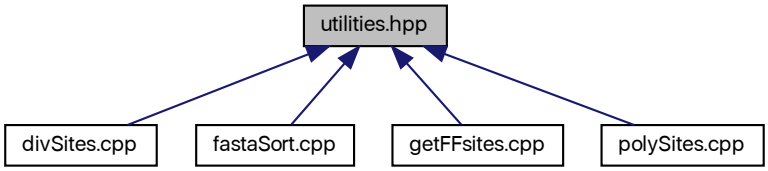
Utility functions.

```
#include <string>
#include <unordered_map>
```

Include dependency graph for utilities.hpp:



This graph shows which files directly or indirectly include this file:



Functions

- void BayesicSpace::parseCL (int &argc, char **argv, unordered_map< char, string > &cli)

  Parse command line flags.

5.11.1 Detailed Description

Utility functions.

Author

Anthony J. Greenberg

Copyright

Version

0.1

5.11.2 Function Documentation

5.11.2.1 parseCL()

```
void BayesicSpace::parseCL (
            int & argc,
            char ** argv,
            unordered_map< char, string > & cli )
```

Parse command line flags.

Parameters

in	*argc*	number of arguments
in	*argv*	array of argument values
out	*cli*	flag values, indexed by flag IDs

Index