Kestrel Manual
v1.0.0

Contents

1 Introduction

1.1 About Kestrel

Kestrel is an alignment-free reference-guided variant caller. Given a reference sequence and sequence reads from an isolate, Kestrel identifies variations between the reference and the isolate. Most reference sequences are contained in one or more FASTA files, and they are long assembled representations of an organisms genome. Sequence reads if an individual, or isolate, are typically contained in one or more FASTQ files.

Kestrel can identify both single nucleotide polymorphism (SNP) and insertion/deletion (indel) variants without relying on sequence reads. Most variant-calling software reads alignments generated by other tools, such as BWA or bowtie. Kestrel reads directly from the FASTQ files and skips the genome alignment step. Although it uses some algorithms based on alignments, Kestrel is alignment-free because it never attempts to align the sequence reads to the reference.

The approach Kestrel takes is to first convert the sequence reads to k-mers, which are short overlapping fragments of a given length (k). For example, the 4-mers in "AACCGG" are "AACC", "ACCG", and "CCGG". By default, Kestrel uses a k-mer size of 31. Because it can communicate directly with KAnalyze, Kestrel can transform the sequence reads very quickly and begin the variant-calling process.

Kestrel has several advantages over traditional alignment-guided variant callers. In some cases, it may be faster because good alignments can take a long time build. Most importantly, it can process variant-dense regions were alignments fail, and it can identify arbitrarily large insertions.

When alignment techniques work correctly, they can produce more statistically-significant results. For Kestrel to function without relying on alignments, some of the original context is lost. For example, alignments can be improved with paired-end reads, but the paired-end context is lost in k-mer space. We recommend using Kestrel as a fast first-pass variant caller or in circumstances where alignments fail or where large insertions are suspected. In some pipelines, both alignment-guided and alignment-free approaches may be used together to find all possible variants.

2 Command Line Usage

Java 1.7 (aka Java 7) must be installed in order to run any commands in this section. Run `java -version` to see your version of Java. If you do not see "java 1.7.0" or a later version, Java must be updated before continuing.

2.1 Java Command

General form:
`java -jar -Xmx3G kestrel.jar <arguments ...>`

This allocates 3 GB of RAM (-Xmx3G).

2.2 Example Usage

2.2.1 Example Usage

`java -jar kestrel.jar count -r ref.fasta sample_1.fastq sample_2.fastq`
Reads reference sequences from ref.fasta and calls variants from the two FASTQ files as one sample.

`java -jar kestrel.jar count -r ref.fasta -i regions.bed sample_1.fastq sample_2.fastq`
Reads reference sequences from ref.fasta, extracts regions from the reference sequences in regions.bed, and calls variants from the two FASTQ files as one sample.

2.3 Command options

2.3.1 Input and Output

Option	Argument	Description	Default
-f --format	FORMAT	Specify the format of input sequence files. Built-in readers are "fastq", "fasta", "fastqgz", "fastagz" for FASTQ and FASTA files and gzipped versions of them, and "ikc" for indexed k-mer count files (a KAnalyze format). The default, "auto", will attempt to determine the file type by its name. This option applies to all input files that follow it on the command line.	auto
-o --out	FILE	Name of the file where output should be directed.	kestrel.table
--stdout		Send output to the screen, STDOUT.	
-m --outfmt	FORMAT	Format of the output. Built-in formats are "table" for a tab-delimited table, and "txt" for text output.	table
--logfile	FILE	Specify the name of a log file.	
--logstdout		Send log messages to STDOUT.	
--logstderr		Send log messages to STDERR.	TRUE
--loglevel	LEVEL	Set the logging level. Valid log levels are "ALL", "TRACE", "DEBUG", "INFO", "WARN", "ERROR", and "OFF".	OFF

--filespersample	N	Automatically divide input files into discrete samples by the number of samples. This may be useful for loading many samples by listing them on the command-line. For example, to load paired-end reads in multiple samples, set this value to "2". Note that Kestrel does not associate files by their name, so related files must be grouped together in the order they are listed. By defaulte, Kestrel loads all files as a single sample.	∞
-s --sample	<NAME>	All input files on the command line following this option are grouped into a single sample with name "NAME". If "NAME" is not specified, then the sample name is derived from the first input file in the sample set. If --filespersample was specified, it is undone and all samples go into this sample set. If --filespersample is specified at some point after this option, it takes precedence and splits files into samples.	
-r --ref	FILE	Specifies the name of the FASTA file containing the reference sequences variants are called against. This option may be used multiple times, and FASTA files may contain any number of records. All FASTA Records must have a unique description.	
-k --ksize	KSIZE	Specify the size of k-mers sequence data is transformed to.	31
--lib	FILE	Load an external library containing extensions for Kestrel, such as a custom output format. FILE may be a JAR file or a directory containing Java class files.	
--liburl	URL	Load a library containing extensions for Kestrel. The library is specified as a URL string, and it can be any URL facility that Java can recognize, locate, and load.	
--charset	CHARSET	Character set encoding of input files. This option specifies the character set of all files following it. The default, "UTF-8", properly handles ASCII files, which is a safe assumption for most files. Latin-1 files with values greater than 127 will not be properly parsed.	UTF-8
--seqfilter --quality	SPEC	Filter sequences as they are read and before k-mers are extracted from them. Some sequence readers can filter or alter reads at runtime. The most common filter is a quality filter where low-quality bases are removed. The filter specification is a filter name followed by a colon (:) and arguments to the filter. If a filter name is not specified, then the "sanger" quality filter is assumed. For example, "sanger:10" and "10" will filter k-mers with any base quality score less than 10. The sequence filter specification is set for all files appearing on the command-line after this option. To turn off filtering once it has been set, files following --noseqfilter will have no filter specification.	
--noseqfilter		Turn off filters for input files following this option.	
--temploc	DIR	Name of the directory where KAnalyze stores temporary files.	Current directory
--mincount	COUNT	A k-mer with a frequency of this value or less is ignored. This keeps the IKC file to a reasonable size by reducing the number of erroneous k-mers from sequencing errors.	5
--minsize		Minimizers group k-mers in the indexed k-mer count (IKC) file generated by Kestrel when reading sequences, and this parameter controls the size of the minimizer.	15

--minmask		K-mers over low-complexity loci may create large minimizer groups, and a minimizer mask may break up these groups.	0x00000000
--memcount		When sequence reads are input, this option will count them in memory instead of an IKC file.	
--nomemcount		When sequence reads are input, generate an IKC file and query k-mer frequencies from it.	TRUE
--rmikc		Remove the indexed k-mer count (IKC) for each sample after kestrel runs.	TRUE
--normikc		Do not remove the indexed k-mer count (IKC) file for each sample.	TRUE
--countrev		Count reverse complement k-mers in region statistics. This should be set for most sequencing protocols.	TRUE
--nocountrev		Do not include the reverse complement of k-mers in read depth estimates. If all sequence reads are in the same orientation as the reference, then this option should be used.	
-i --interval	FILE	Reads a file of intervals defining the regions over the reference sequences where variants should be detected. If no intervals are specified, variants are detected over the full length of each reference sequence. The file type is determined by the file name, such as "intervals.bed".	
--flank	LENGTH	When reading an interval from a reference, extract sequences beyond the boundaries of the interval to assist active region detection. Variants within the flanks are discarded.	$k \cdot 3.5$
--autoflank		The flank length is automatically chosen by the k-mer size.	$k \cdot 3.5$
--byreference		If variant call regions were defined, variant call locations are relative to the reference sequence and not the region.	TRUE
--byregion		If variant call regions were defined, variant call locations are relative to each region instead of the reference sequence.	
--rmrefdesc		When set, remove the description from reference sequence names. The descirption is everything that occurs after the first whitespace character. FASTA files often have a sequence name and a long description separated by whitespace. This option ensures that the sequence name matches in the FASTA and an interval file, if used.	TRUE
--normrefdesc		Use the full sequence name as it appears in the reference sequence file. FASTA files often include a description after the sequence name, and with this option, it becomes part of the full sequence name. If using an interval file, the full sequence name and description must match the sequence file.	
--revregion		When set, reverse complement reference regions that occur on the negative strand, and all variant calls are relative to the reverse-complemented sequence. Only itervals defined with on the negative strand are altered.	
--norevregion		When set, regions variants are called on are always in the same orientation as the reference sequence. The stranded-ness of defined intervals is ignored.	TRUE

2.4 Active Region Detection

Option	Argument	Description	Default
--mindiff	DIFF	Set the minimum k-mer count difference for identifying active regions. The difference threshold determined by the difference quantile will never be less than this value.	5
--diffq	DIFFQ	If set to a value greater than 0.0, then the k-mer count difference between two k-mers that triggers an active region scan is found dynamically. The difference in k-mer counts between each pair of neighboring k-mers over an uncorrected reference region is found, and this quantile of is computed over those differences. The default value of 0.90 means that at most 10% of the k-mer count differences will be high enough. If this computed value is less than the minimum k-mer count difference (--mindiff), then that minimum is the difference threshold. This value may not be 1.0 or greater, and it may not be negative. If 0.0, the minimum count difference is always the minimum threshold (--mindiff).	0.90
--peakscan	LENGTH	Reference regions with sequence homology in other regions of the genome may contain k-mers with artificially high frequencies from adding counts for k-mers that appear in those regions. This causes a peak in the k-mer frequencies over the reference, and it can trigger an erroneous active-region scan for variants. Kestrel will scan forward this number of k-mers looking for a peak in the k-mer frequencies. If the counts drop back down to the original range, the active region scan is stopped. This keeps Kestrel from erroneously searching large regions of the reference. Setting this value to 0 disables peak detection.	7
--scanlimitfactor	FACTOR	Set a limit on how long an active region may be. This is computed by multiplying the k-mer size by this factor and adding the maximum length of a gap. The computed limit will be adjusted so that active regions are at least large enough to capture a SNP in cases where the maximum gap length is 0. Setting this to a low value or m̈inẅill set the limit so that it is just large enough to catch SNPs and deletions, but it will miss large deletions if another variant is within the k-mer size window. Setting this to a high value or "m̈axl̈ifts the restrictions on active region lengths, and this may cause the program to take an excessive amount of time and memory trying to solve arbitrarily long active regions.	5.0
--decaymin	FACTOR	Set the minimum value (asymptotic lower bound) of the exponential decay function used in active region detection as a proportion of the anchor k-mer count. If this value is 0.0, k-mer count recovery threshold may decline to 1. If this value is 1.0, the decay function is not used and the detector falls back to finding a k-mer with a count within the difference threshold of the anchor k-mer count.	0.55
--alpha	ALPHA	Set the exponential decay alpha, which controls how quickly the recovery threshold declines to its minimum value (see --decaymin) in an active region search. Alpha is defined as the rate of decay for every k bases. At k bases from the left anchor, the threshold will have declined to $\alpha\cdot$ range. At every k bases, the threshold will continue to decline at this rate ($\alpha^n\cdot$ range).	0.80

--anchorboth		Active regions may not go to the ends of the references. This reduces false-calls from noise, but it can miss variant calls near the ends.	TRUE
--noanchorboth		Active regions ma go to the ends of the references. This may call variants near the ends, but it may also result in false calls.	

2.5 Haplotype Reconstruction

Option	Argument	Description	Default
-w --weight	VECTOR	Set the alignment weights as a comma-separated list of values. The order of weights is match, mismatch, gap-open, gap-extend, and initial score. If values are blank or the list has fewer than 5 elements, the missing values are assigned their default weight. Each value is a floating-point number, and it may be represented in exponential form (e.g. 1.0e2) or as an integer in hexadecimal or octal format. Optionally, the list may be surrounded by parenthesis or braces (angle, square, or curly). If the initial score is empty or missing, it defaults to the k-mer size multiplied by the match score.	"10, -10, -40, -4, 0"
--maxalignstates		Set the maximum number of alignment states. When haplotype assembly branches into more than one possible sequence, the state of one is saved while another is built. When the maximum number of saved states reaches this value, the least likely one is discarded.	15

2.6 Variant Calling

Option	Argument	Description	Default
--ambiregions		Call variants in active regions that contain ambiguous bases.	TRUE
-- noambigregions		Discard all active regions that contain at least one ambiguous base.	
--ambivar		Call variants at where the reference contains an ambiguous base.	TRUE
--noambivar		Discard all variant calls over ambiguous reference bases.	
--varfilter	SPEC	Add a variant filter specification. The argument should be the name of the filter, a colon, and the filter arguments. The correct filter is loaded by name and the filter arguments are passed to it.	

2.6.1 Other

Option	Argument	Description	Default
--free		Free resources between processing samples. This may reduce the memory footprint of Kestrel, but it may force expensive resources to be recreated and impact performance.	

--nofree		Retain resources between samples. This may use more memory, but it will avoid re-creating expensive resources between samples.	TRUE

3 The Kestrel Process

Kestrel is not like other variant callers, and understanding the process is important for correctly applying it. This section discusses the process of translating sequence reads to results.

3.1 Overview

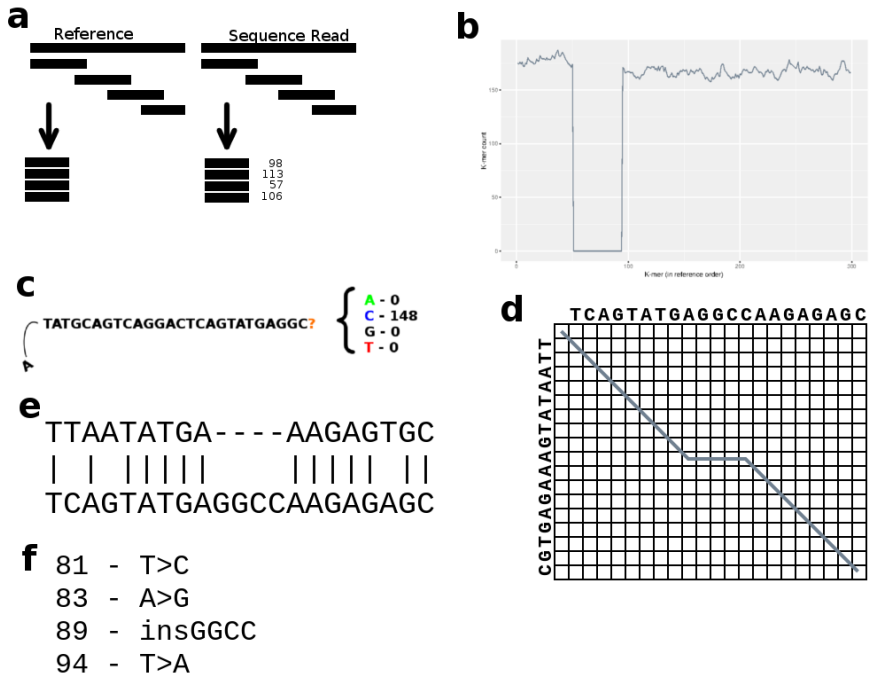

Figure 1: Overview of the Kestrel process from sequence data to variant call. (**a**) The reference sequence and the sequence reads are converted to k-mers. The k-mers of the sequence reads are counted, but the k-mers of the reference are left in order. (**b**) K-mer frequencies from the sequence reads (vertical axis) are assigned to the k-mers of the reference (horizontal axis). A decline and recovery of the frequencies bound an active region where one or more variants are present. (**c**) Starting from the left anchor k-mer (last k-mer with a high frequency), the first base is removed, each possible base is appended, and the base that recovers the k-mer frequency is appended to the haplotype. (**d**) A modified alignment algorithm tracks haplotype reconstruction and terminates the process when an optimal alignment is reached. (**e**) This algorithm yields an alignment of the reference sequence and haplotype within the active region. (**f**) Variant calls are extracted from the alignment.

Like other variant calling approaches, Kestrel reads sequence data, compares it to a reference, and determines how the sequence data varies from the reference. The standard approach is to map the sequences to the reference and search for differences in that map. Where the sequences vary significantly from the reference, such as large insertions or dense SNPs, the read mapping step can fail. Since Kestrel does not map the sequence reads, it can recover these variations.

Instead of read mapping, sequence data is translated into a set of k-mer frequencies, and this supports the variant calls. K-mers is a short overlapping fragment of uniform length, k, extracted from a sequence. For example, the sequence "ACGTAC" in 4-mers is "ACGT", "CGTA", and "GTAC". These k-mers are extracted from all sequence reads, and the frequency of each unique k-mer is counted and stored in an indexed k-mer count (IKC) file on disk (**Fig. 1a**). In real data, the k-mer size should generally between 31 and 48. Kestrel manages k-mer counting through the KAnalyze Audano and Vannberg (2014) application programming interface (API).

The reference sequence is also converted to k-mers of the same size, but these left in order (**Fig. 1a**). Each reference k-mer and its reverse complement are queried from the IKC file, which gives an array of frequencies for each reference k-mer as it was found in the sample. This array is scanned for a region of

k-mers where frequency drops and then recovers (**Fig. 1b**). This defines an "active region" where the k-mers from the reference and the k-mers from the sample appear to be different.

The k-mer frequency database is then used as evidence to reconstruct the sequence over the active region. This reconstructed sequence is called a "haplotype", and each haplotype covers the whole active region. Note that more than one variant may be contained in a haplotype.

The high-frequency k-mers immediately adjacent to the low frequency k-mers in the active region are called "anchor k-mers", and there is one on each end of the active region. These anchor k-mers guide the haplotype reconstruction process, and they are part of the active region.

Haplotype reconstruction starts with the left anchor k-mer, which has a high frequency and is assumed to match the reference. The first base of the k-mer is removed to create a $(k-1)$-mer. Each of the four bases are appended to the $(k-1)$-mer in turn to see which one brings the k-mer frequency back up. This base is then added to the haplotype (**Fig. 1c**), and this process is repeated on the new k-mer. If more than one k-mer with a high count is found during this step, the haplotype splits and multiple haplotypes are constructed.

Reconstruction must eventually cease when the haplotype is acceptable or when it cannot be an acceptable solution, so a modified Smith-waterman Smith and Waterman (1981) alignment algorithm guides this step (**Fig. 1d**). This alignment forces the haplotype and the active region to align on the left end, but it allows the haplotype to extend. When an optimal alignment is found, reconstruction is stopped.

From the alignment (**Fig. 1e**), it is trivial to identify the variants (**Fig. 1f**). The evidence for each variant is added from all the haplotypes that support it.

3.2 Input

Kestrel can input sequence reads or an indexed k-mer count (IKC) file. If sequence reads are input, then k-mers are counted and output to an IKC file.

By default, all input files are part of the same sample. If multiple samples are given, then variants are called on each one independently. The command-line options in **Section 2.3.1** contain options for breaking input into discrete samples.

3.3 Intervals

3.4 K-mer Database

3.5 Active Region Detection

3.6 Haplotype Reconstruction

3.7 Variant Calling

4 API

Kestrel is not only a command line program, it is also an API (Application Programming Interface) that can be driven from other programs. The CLI (Command Line Interface) itself is a simple application that calls the API to carry out tasks.

Kestrel is a Java program, so the API is implemented as a set of Java classes with well documented interfaces. Any Java program can directly call the API classes. Other programs can use technologies such as JNI (Java Native Interfaces) to drive the API from native code, such as C or C++.

Lastly, the command line itself can be used to drive the program from a script or batch file. Because Kestrel always terminates with a well-defined return code, scripts can easily execute the CLI and check for errors. For a list of the return codes, see section 6.3.

The remainder of this section outlines the structure and capabilities of the API. It is assumed that a reader of this section has at least a fundamental understanding of Java.

4.1 Javadoc Documentation

The Kestrel API is fully documented with Javadoc comments. Every class and class member (method and field), regardless of scope, has a Javadoc comment. All method arguments, return values, and exceptions are contained in the method's Javadoc comment. For any method arguments that are objects, the comment must disclose what happens when `null` values are received for that field (either by the argument's comment, or the comment on the exception if one is thrown). All exceptions, whether or not they are runtime exceptions, must be documented along with the conditions that cause them to be thrown. Assertion errors are not documented. Any deviations from these rules should be reported as a program bug.

Javadoc comments can be converted into a set of HTML pages that document the code. In fact, the Java API itself uses Javadoc, so most Java programmers are familiar with the format of the HTML pages. The Kestrel build system will create the Javadoc pages for two different levels: api and full. The API level contains only public and protected members, while the full documentation contains everything. For a programmer interested in using the API or creating a custom component, the API level documentation is probably sufficient. The full documentation is intended for maintenance programmers.

For more information on building Javadoc HTML pages from source, see section 6.4.1.

4.2 Project Organization

4.3 Utility Classes

Kestrel has several utility classes for shared functionality. Most of the classes are used by multiple components or modules. These classes may also be useful for other applications.

All of these classes are found in package `edu.gatech.kestrel.util`

4.3.1 InfoUtil

5 Implementation Details

This section contains important implementation details for those wishing to extend or maintain Kestrel.

5.1 Condition Reporting System

TODO: Fill in this section.

6 Supplementary Information

6.1 Input File Formats

6.1.1 FASTA

FASTA files are compatible with the FASTA format acceptable by NCBI BLAST[1]. The only exception is that the reader permits blank lines within the sequence because all blank lines are ignored by the parser.

A single-line description beginning with '¿' precedes each sequence record. The sequence record may be on a single line or split over any number of lines. There are no restrictions on the line length.

FASTA files are not checked by the reader. The sequence string may contain any characters as long a line does not begin with '¿'. The k-mer component that translates sequences skips over all unrecognized characters. Checking sequences while reading is omitted for performance reasons.

Standard line break characters are recognized: LF (Linux/Unix/MacOS), or CRLF (Windows). The file may or may not end with a newline character.

6.1.2 FASTAGZ

A GZIP compressed FASTA file. The uncompressed file contents must follow the FASTA format defined in Section 6.1.1.

6.1.3 FASTQ

FASTQ files must be compatible with the format published in Nucleic Acid Reseach[2]. The first line of a record begins with '@'. The sequence may be split over multiple lines. A line beginning with '+' will separate the sequence from the quality scores. Quality scores are read line-by-line until the number of quality score characters is the same as the number of sequence characters read for any given record. The quality score characters are discarded (this reader does not enforce proper scores).

This reader permits blank lines between records, but not within records.

Standard line break characters are recognized: LF (Linux/Unix/MacOS), or CRLF (Windows). The file may or may not end with a newline character.

6.1.4 FASTQGZ

A GZIP compressed FASTQ file. The uncompressed file contents must follow the FASTQ format defined in Section 6.1.3.

6.1.5 RAW

Raw sequence files are the simplest record type. A sequence may be split over any number of lines. Blank lines separate records.

Standard line break characters are recognized: LF (Linux/Unix/MacOS), or CRLF (Windows). The file may or may not end with a newline character.

[1] http://www.ncbi.nlm.nih.gov/BLAST/blastcgihelp.shtml
[2] http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2847217/

6.2 Kmer Output Formats

K-mers may be output in one of several formats. The most natural is the sequence format, which a sequence of k bases, each A, C, T, or G. Note that if RNA sequences are read, U is converted to T on output. This is because T and U are represented the same internally.

Two other representations, decimal and hexadecimal, take a little more explanation. Internally, k-mers are represented as integers. Converting the sequences to integers makes processing and storing the k-mers fast and efficient.

To understand the k-mer nucletide sequence to integer conversion, the nucleotide sequence can be thought of a base-4 numbering system where A is 0, C is 1, G is 2, and T is 3. Each base can be conveniently represented as a two-bit number where A = 00, C = 01, G = 10, and T = 11. A k-mer sequence is then a sequence of these letters. For example, the 8-mer CCAGTCGT is 0101001011011011.

Note that k-mers are given a natural order where lower numbers come before higher numbers. For 8-mers, the first possible k-mer is AAAAAAAA, and the last possible k-mer is TTTTTTTT. This numeric ordering can be taken advantage of in many ways.

Java's long integer is a 64-bit 2's complement integer. Each k-mer is is two bits in length. In order to preserve the natural ordering, we chose not to include negative numbers, which is any 2's complement integer that begins with 1. Excluding the first bit, which is always 0, a 64-bit integer can represent any k-mer size 1 to 31. Therefore, the maximum k-mer size is currently 31.

Binary numbers are conveniently represented in hexadecimal. K-mers can also be written as hexadecimal integers, which affords them a compact representation. For example, the same 8-mer as above, CCAGTCGT, can be represented as 0x52DB.

The global property, *kanalyze.outfmt*, sets one of the following output options. If the property is not set, the default sequence option is set. For convenience, most modules have a shorter command line option to set this property.

6.2.1 Sequence Format

Tab delimited file of k-mers and counts. K-mers are written as a string of A, C, G, and T. Each k-mer has k letters. Note that if RNA sequences were read, U will be converted to T when sequences are output in this format.

6.2.2 Decimal Format

Tab delimited file of k-mers and counts using the base-10 decimal representation of k-mers. For more information about how k-mer sequences are represented as integers, see the above section 6.2.

6.2.3 Hexadecimal Format

Tab delimited file of k-mers and counts using the base-16 decimal representation of k-mers. For more information about how k-mer sequences are represented as integers, see the above section 6.2.

6.2.4 FASTA Format

FASTA representation of k-mers counts. Each FASTA record is a single k-mer and its count. The description line is the k-mer count and the sequence is the k-mer sequence.

6.3 Command Line Return Codes

The KAnalyze user interface always returns a well-defined code. When executed from a script environment, this return code can easily be checked to see if KAnalyze completed normally or not. Each return code is defined in this section.

In the following list of return codes, the numeric code is listed. For API users, the constant defined in `edu.gatech.kanalyze.Constants` is also listed.

0 (ERR_NONE) The program terminated normally. All k-mers were successfully processed without error, or the help option was invoked.

1 (ERR_USAGE) Command line arguments were incomplete, improperly formatted, or required arguments were missing.

2 (ERR_IO) An I/O (input/output) error occurred reading or writing data. This error is normally returned for file I/O errors.

3 ERR_SECURITY A security error, such as permissions denied, occurred.

4 (ERR_FILENOTFOUND) A required or specified file was not found.

5 (ERR_DATAFORMAT) Data in an input file is improperly formatted.

6 (ERR_ANALYSIS) Some un-recoverable error occurred during analysis.

7 (ERR_INTERRUPTED) A program thread was interrupted while it was running. This would normally be returned if the program is terminated before it completes.

98 (ERR_ABORT) The program or a process was terminated before it completed. When the action is requested or expected, this should be returned in lieu of ERR_INTERRUPTED even if interrupting threads was necessary.

99 (ERR_SYSTEM) Some serious unrecoverable system error occurred. These errors are almost certainly program bugs. Some other unrecoverable errors, such as running out of memory, could return this code.

6.4 Building From Source

The source is distributed with an Apache Ant build file with multiple targets. To build these, Apache Ant must be installed.

To run a target, enter `ant <target>`. Multiple targets may be supplied, for example `ant clean package`.

The most commonly used targets are "clean", "compile", and "package". The clean removes all temporary files. It is not often necessary, but it should be executed before building a package for testing to ensure no artifacts are left behind from previous builds. The compile target compiles the class files and does nothing else. The package target generates the JAR file and distributable packages.

The "doc.javadoc" target generates the Javadoc pages from Kestrel source comments. See Section 6.4.1 for more information about these pages. "doc.manual" generates this manual as a PDF file.

Other targets are called by the targets described above and are not often called directly.

For a full list of targets and brief descriptions, enter `ant -projecthelp`.

6.4.1 Building Javadoc Pages

Javadoc is a very powerful tool for documenting APIs written in Java. Section 4.1 introduces the concept and the KAnalyze rules governing these comments.

Javadoc comments begin with "/**", and end with "*/". In Kestrel, these comments appear before every class, method, and field. The comment starts with a description of what the element does. For methods, it documents parameters and the conditions under which all exceptions are thrown. Full documentation for writing these comments can be found online[3].

The ant build system (Section 6.4) generates web pages from these comments. The "doc.javadoc" target builds two sets of pages. One, the API documentation, documents all elements available on the API. This target is intended for developers who are extending or using the API. The other, FULL documentation,

[3] http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html

documents everything including private members that are not available to the API. This is meant for Kestrel maintenance programmers.

After running the javadoc Ant target, the API documentation can be access by loading “build/doc/javadoc/api/index.html in a browser (relative to the project root). The FULL documentation can be access by loading “build/doc/javadoc/full/inde in a browser (relative to the project root).

7 License

7.1 Documetation

This document is licensed under the GNU Free Documentation License 1.3 or later. The file "COPYING.DOC" contains the text of this license. If you did not receive the file in the source distribution, please contact us or the GNU website for a copy.

7.2 KAnalyze Software

The Kestrel software is licensed under the GNU Lesser General Public License version 3 or later. The file "COPYING" contains the text of the GNU GPL, and the file "COPYING.LESSER" contains the GNU LGPL extension to the GPL. If you did not receive the file in the source distribution, please contact us or the GNU website for a copy.

References

P. Audano and F. Vannberg. KAnalyze: a fast versatile pipelined K-mer toolkit. *Bioinformatics (Oxford, England)*, 30(14):2070–2, July 2014. ISSN 1367-4811. doi: 10.1093/bioinformatics/btu152. URL `http://bioinformatics.oxfordjournals.org/content/30/14/2070.full`.

T. Smith and M. Waterman. Identification of common molecular subsequences. *Molecular Biology*, 147(1):195–197, 1981. ISSN 0022-2836. URL `http://www.ncbi.nlm.nih.gov/pubmed/7265238`.