SMRT® Tools Reference Guide

Introduction

This document describes the command-line tools included with SMRT Link v5.1.0. These tools are for use by bioinformaticians working with secondary analysis results.

- The command-line tools are located in the `$SMRT_ROOT/smrtlink/smrtcmds/bin` subdirectory.

Installation

The command-line tools are installed as an integral component of the SMRT Link software. For installation details, see **SMRT Link Software Installation (v5.1.0)**.

- To install **only** the command-line tools, use the `--smrttools-only` option with the installation command, whether for a new installation or an upgrade. Examples:

```
smrtlink-*.run --rootdir smrtlink --smrttools-only
smrtlink-*.run --rootdir smrtlink --smrttools-only --upgrade
```

Pacific Biosciences Command-Line Tools

Following is information on the Pacific Biosciences-supplied command-line tools included in the installation. Third-party tools installed are described at the end of the document.

arrow

This is the `variantCaller` tool with the consensus algorithm set to `arrow`. See "variantCaller" on page 86 for details.

bam2bam

The `bam2bam` tool reprocesses, and optionally converts, BAM files from one convention to another. For example, a BAM file containing HQ regions could be processed, adapter hits and barcodes identified, and a new subreads BAM file produced.

Note: This tool is **deprecated** for use with barcoded data; use the **Demultiplex Barcodes** tool instead. See "Demultiplex Barcodes" on page 23 for details.

This tool is useful where `PostPrimary` on the instrument was used incorrectly, such as performing spike-in control filtering on a local computer because the filter controls were not specified on the instrument

- Both production and pulse BAM files can be processed.
- "Scraps" BAM files are always required to reconstitute the ZMW reads internally. Conversely, "scraps" BAM files will be output.

- ZMW reads are **not** allowed as input, due to the missing HQ-region annotations.
- Input read convention is determined from the `READTYPE` annotation in the `@RG::DS` tags of the input BAM files.

`bam2bam` is installed on every Sequel® System and is shipped with SMRT® Analysis.

Usage

```
-o outputPrefix [options] input.(subreads|hqregion).bam
input.(scraps).bam
```

Example

```
bam2bam in.subreads.bam in.scraps.bam -o out --barcodes bc.fasta
```

Required Parameter

Required	Description
`-o STRING`	Prefix of the output file names.

Optional Parameters

Options	Description
`-j INT`	Number of threads for parallel ZMW processing.
`-b INT`	Number of threads for parallel BAM compression.
`--silent`	Do **not** display progress output.

BAM Conventions

Options	Description
`--zmw`	Create a ZMW read.
`--hqregion`	Output `*.hqregions.bam` and `*.scraps.bam`.

Parameter for Finding Adapters

Option	Description
`--adapters=adapterSequences.fasta`	The file name of the adapter sequence(s). This specifies that adapter-calling should be run.

Parameters for Finding Barcodes

Options	Description
`--barcodes=barcodeSequences.fasta`	Specify a FASTA file of input barcode sequences to enable finding and labeling barcodes.
`--hotStartMode`	Enable searching for barcodes at the beginning of a read if no adapters are found. (Default = `False`)
`--scoreFirst`	Alternate name for `--hotStartMode`. (Default = `False`)

Options	Description
`--hotStartLength=INT`	Number of bases used to find the hot start barcode if the adapter is missing. (Default = `100`)
`--maxAdapters=INT`	Number of adapters used to determine the barcodes. (Default = `4`)
`--scoreMode=STRING`	Barcode-calling mode. `symmetric`: Each barcode sequence identifies a single bin for demultiplexing reads. `asymmetric`: Barcode sequences are different on either end of an insert present in a SMRTbell® template. (Default = `symmetric`)

Parameters for Filtering Control Sequences

Options	Description
`--controls=controlSequences.fasta`	Enables control sequence-filtering.
`--maxControls=INT`	Number of subreads used to determine if a ZMW is a control. (Default = `3`)

Additional Output Read Types

Options	Description
`--fasta`	Output `fasta.gz`.
`--fastq`	Output `fastq.gz`.
`--noBam`	Do **not** produce BAM outputs.

Parameters for Fine Tuning

Options	Description
`--minAdapterScore=int`	Minimum score for an adapter.
`--minPolyLength=INT`	Minimum ZMW real length. (Default = `1`)
`--minSubLength=INT`	Minimum subread length. (Default = `1`)
`--fullHQ`	Disable HQRF - the entire ZMW read will be deemed "HQ".

Examples

To use a new adapter-finding algorithm on an older data set:

```
$ bam2bam --adapter adapters.fasta                  \
          -o movieName.newAdapters                  \
          movieName.subreads.bam movieName.scraps.bam
```

To convert subreads plus scraps to ZMW reads:

```
$ bam2bam --zmw                                     \
          -o movieName.stitched                     \
          movieName.subreads.bam movieName.scraps.bam
```

To output stitched ZMW reads additionally in `FASTA.GZ` format (a `*.zmw.BAM` file is automatically created):

```
$ bam2bam --zmw                                     \
          --fasta                                   \
          -o movieName.stitched                     \
          movieName.subreads.bam movieName.scraps.bam
```

To convert subreads plus scraps to hqregions+scraps:

```
$ bam2bam --hqregion                                \
          -o movieName.stitchedHQ                   \
          movieName.subreads.bam movieName.scraps.bam
```

To output `hqregions` additionally in `FASTA.GZ` format (a `*.hqregions.BAM` file is automatically created):

```
$ bam2bam --hqregion                                \
          --fasta                                   \
          -o movieName.stitched                     \
          movieName.subreads.bam movieName.scraps.bam
```

To output subreads in `FASTA.GZ` format **only**:

```
$ bam2bam --nobam                                   \
          --fasta                                   \
          -o movieName.new                          \
          movieName.subreads.bam movieName.scraps.bam
```

To convert hqregions plus scraps to subreads plus scraps with adapter and barcodes:

```
$ bam2bam --barcodes barcodes.fasta                  \
          --adapter adapters.fasta                   \
          -o movieName.newVersion                    \
          movieName.hqregions.bam movieName.scraps.bam
```

To perform a sanity check to ensure that the output is the same as the input, and add a new BAM header entry with the `bam2bam` version:

```
$ bam2bam -o movieName.sanity                       \
          movieName.subreads.bam movieName.scraps.bam
$ samtools view movieName.subreads.bam > a1
$ samtools view movieName.sanity.subreads.bam > b1
$ diff a1 b1
$ samtools view movieName.scraps.bam > a1
$ samtools view movieName.sanity.scraps.bam > b1
$ diff a1 b1
$ rm a1 b1
```

To perform spike-in control filtering on a local computer because the filter controls were not specified on the instrument:

```
$ bam2bam --controls control_orig.fasta               \
          -o movieName.control_orig                   \
          movieName.subreads.bam movieName.scraps.bam
```

To use a better reference for the spike-in controls:

```
$ bam2bam --controls control_better.fasta             \
          -o movieName.control_better                 \
          movieName.subreads.bam movieName.scraps.bam
```

To perform a complete analysis from scratch, as the primary analysis software was released with a new set of improved algorithms: (**Note**: Only HQ regions **cannot** be computed from scratch)

```
$ bam2bam --barcodes barcodes.fasta                   \
          --adapter adapters.fasta                    \
          --controls control.fasta                    \
          -o movieName.newPPAVersion                  \
          movieName.subreads.bam movieName.scraps.bam
```

To treat the complete ZMW read as an HQ region and perform adapter-finding:

```
$ bam2bam --fullHQ                                    \
          --adapter adapters.fasta                    \
          -o movieName.fullhq                         \
          movieName.subreads.bam movieName.scraps.bam
```

bam2fasta/ bam2fastq

The `bam2fastx` tools convert PacBio® BAM files into gzipped FASTA and FASTQ files, including demultiplexing of barcoded data.

Usage

Both tools have an identical interface and take BAM and/or Data Set files as input.

Examples

```
bam2fasta -o projectName m54008_160330_053509.subreads.bam

bam2fastq -o myEcoliRuns m54008_160330_053509.subreads.bam
m54008_160331_235636.subreads.bam

bam2fasta -o myHumanGenomem54012_160401_000001.subreadset.xml
```

Input Files

- One or more `*.bam` files
- `*.subreadset.xml` file (Data Set file)

Output Files

- `*.fasta.gz`
- `*.fastq.gz`

bax2bam

The `bax2bam` tool converts the legacy PacBio basecall format (`bax.h5`) into the BAM basecall format.

Usage

```
bax2bam [options] <input files...>
```

Options

Options	Description
`-h`, `--help`	Display help information and exits.
`--version`	Displays program version number and exits

Pulse feature options

These options configure pulse features in the output BAM. Supported features include:

Pulse Feature	BAM Tag	Default
`DeletionQV`	`dq`	Y
`DeletionTag`	`dt`	Y
`InsertionQV`	`iq`	Y
`IPD`	`ip`	Y
`PulseWidth`	`pw`	N
`MergeQV`	`mq`	Y
`SubstitutionQV`	`sq`	Y
`SubstitutionTag`	`st`	N

If the Pulse Feature option is used, then **only** those features listed will be included, regardless of the default state.

- `--pulsefeatures=STRING` (Comma-separated list of desired pulse features, using the names in the table above.)
- `--losslessframes` (Store full, 16-bit IPD/PulseWidth data, instead of (default) downsampled, 8-bit encoding.)

Input Files

- `movie.1.bax.h5`, `movie.2.bax.h5` ... (**Note**: Input files should be from the same movie.)
- `--xml=STRING` (Data Set XML file containing a list of movie names.)
- `-f STRING`, `--fofn=STRING` (File-of-file-names containing a list of input files.)

Output Files

- `-o STRING` (Prefix of output file names. The movie name will be used if no prefix is provided.)
- `--output-xml=STRING` (Explicit output XML name. If **not** provided using this option, `bax2bam` will use the `-o` prefix (`<prefix>.dataset.xml`). If that is not specified either, the output XML file name will be `<moviename>.dataset.xml`)
- Output read types: (**Note**: These types are mutually exclusive.)
  - `--subread:` Output subreads (Default)

- `--hqregion`: Output HQ regions
- `--polymeraseread`: Output full polymerase read
- `--ccs`: Output CCS sequences

- Output BAM file type:
  - `--internal` Output BAMs in internal mode. Currently this indicates that non-sequencing ZMWs should be included in the output scraps BAM file, if applicable.

Example

Assuming your original file is named `mydata.bas.h5`, you can produce a file `mynewbam.subreads.bam` using the following command:

```
bax2bam -o mynewbam mydata.1.bax.h5 mydata.2.bax.h5 mydata.3.bax.h5
```

blasr

The `blasr` tool aligns long reads against a reference sequence, possibly a multi-contig reference.

`blasr` maps reads to genomes by finding the highest scoring local alignment or set of local alignments between the read and the genome. The initial set of candidate alignments is found by querying a rapidly-searched precomputed index of the reference genome, and then refining until only high scoring alignments are kept. The base assignment in alignments is optimized and scored using all available quality information, such as insertion and deletion quality values.

Because alignment approximates an exhaustive search, alignment significance is computed by comparing optimal alignment score to the distribution of all other significant alignment scores.

Usage

```
blasr {subreads|ccs}.bam genome.fasta --bam --out aligned.bam [--options]

blasr {subreadset|consensusreadset}.xml genome.fasta --bam --out aligned.bam [--
options]

blasr reads.fasta genome.fasta [--options]
```

Input Files

- `{subreads|ccs}.bam` is in PacBio BAM format, which is the native Sequel System output format of SMRT reads. PacBio BAM files carry rich quality information (such as insertion, deletion, and substitution quality values) needed for mapping, consensus calling and variant detection. For the PacBio BAM format specifications, see **http://pacbiofileformats.readthedocs.io/en/3.0/BAM.html**.
- `{subreadset|consensusreadset}.xml` is in PacBio DataSet format. For the PacBio DataSet format specifications, see: **http://pacbiofileformats.readthedocs.io/en/3.0/DataSet.html**.
- `reads.fasta`: A multi-FASTA file of reads. While any FASTA file is valid input, `bam` or `dataset` files are preferable as they contain more rich quality value information.

- `genome.fasta:` A FASTA file to which reads should map, usually containing reference sequences.

Output Files

- `aligned.bam:` The pairwise alignments for each read, in PacBio BAM format.

Input Options

Options	Description
`--sa suffixArrayFile`	Use the suffix array `sa` for detecting matches between the reads and the reference. (The suffix array is prepared by the `sawriter` program.)
`--ctab tab`	A table of tuple counts used to estimate match significance, created by `printTupleCountTable`. While it is quick to generate on the fly, if there are many invocations of `blasr`, it is useful to precompute the `ctab`.
`--regionTable table`	A read-region table in HDF format for masking portions of reads. This may be a single table if there is just one input file, or a `fofn` (file-of-file names). When a region table is specified, any region table inside the `reads.plx.h5` or `reads.bax.h5` files is ignored. **Note**: This option works **only** with PacBio RS II HDF5 files.
`--noSplitSubreads`	Do not split subreads at adapters. This is typically only useful when the genome in an unrolled version of a known template, and contains template-adapter-reverse-template sequences. (Default = `False`)

Options for Aligning Output

Options	Description
`--bestn n`	Provide the top `n` alignments for the hit policy to select from. (Default = `10`)
`--sam`	Write output in SAM format.
`--bam`	Write output in PacBio BAM format.
`--clipping`	Use no/hard/soft clipping for SAM output. (Default = `none`)
`--out file`	Write output to `file`. (Default = `terminal`)
`--unaligned file`	Output reads that are not aligned to `file`.
`--m t`	If **not** printing SAM, modify the output of the alignment. • t=0: Print blast-like output with |'s connecting matched nucleotides. • 1: Print only a summary: Score and position. • 2: Print in `Compare.xml` format. • 3: Print in vulgar format (**Deprecated**). • 4: Print a longer tabular version of the alignment. • 5: Print in a machine-parsable format that is read by `compareSequences.py`.
`--noSortRefinedAlignments`	Once candidate alignments are generated and scored via sparse dynamic programming, they are rescored using local alignment that accounts for different error profiles. Resorting based on the local alignment may change the order in which the hits are returned. (Default = `False`)
`--allowAdjacentIndels`	When specified, adjacent insertion or deletions are allowed. Otherwise, adjacent insertion and deletions are merged into one operation. Using quality values to guide pairwise alignments may dictate that the higher probability alignment contains adjacent insertions or deletions. Tools such as GATK do **not** permit this and so they are not reported by default.
`--header`	Print a header as the first line of the output file describing the contents of each column.

Options	Description
`--titleTable tab`	Build a table of reference sequence titles. The reference sequences are enumerated by row, `0,1,...`. The reference index is printed in alignment results rather than the full reference name. This makes output concise, particularly when very verbose titles exist in reference names. (Default = `NULL`)
`--minPctIdentity p`	Only report alignments if they are greater than `p` percent identity. (Default = `0`)
`--holeNumbers LIST`	When specified, only align reads whose ZMW hole numbers are in `LIST`. `LIST` is a comma-delimited string of ranges, such as `1,2,3,10-13`. This option **only** works when reads are in base or pulse h5 format.
`--hitPolicy policy`	Specifies how `blasr` treats multiple hits: • `all`: Reports **all** alignments. • `allbest`: Reports all equally top-scoring alignments. • `random`: Reports a single random alignment. • `randombest`: Reports a single random alignment from multiple equally top-scoring alignments. • `leftmost:` Reports an alignment which has the best alignment score and has the smallest mapping coordinates in any reference.

Options for Anchoring Alignment Regions

- These options will have the greatest effects on speed and sensitivity.

Options	Description
`--minMatch m`	Minimum seed length. A higher value will speed up alignment, but decrease sensitivity. (Default = `12`)
`--maxMatch m` `--maxLCPLength m`	Stop mapping a read to the genome when the LCP length reaches `m`. This is useful when the query is part of the reference, for example when constructing pairwise alignments for *de novo* assembly. (Both options work the same.)
`--maxAnchorsPerPosition m`	Do **not** add anchors from a position if it matches to more than `m` locations in the target.
`--advanceExactMatches E`	Another trick for speeding up alignments with match `-E` fewer anchors. Rather than finding anchors between the read and the genome at every position in the read, when an anchor is found at position `i` in a read of length `L`, the next position in a read to find an anchor is at `i+L-E`. Use this when aligning already assembled contigs. (Default = `0`)
`--nCandidates n`	Keep up to `n` candidates for the best alignment. A large value will slow mapping as the slower dynamic programming steps are applied to more clusters of anchors - this can be a rate-limiting step when reads are very long. (Default = `10`)
`--concordant`	Map all subreads of a ZMW (hole) to where the longest full pass subread of the ZMW aligned to. This requires using the region table and hq regions. This option **only** works when reads are in base or pulse h5 format. (Default = `False`)

Options for Refining Hits

Options	Description
`--sdpTupleSize K`	Use matches of length `K` to speed dynamic programming alignments. This controls accuracy of assigning gaps in pairwise alignments once a mapping has been found, rather than mapping sensitivity itself. (Default = `11`)

Options	Description

`--scoreMatrix "score matrix string"`	Specify an alternative score matrix for scoring FASTA reads. The matrix is in the format
	`  ACGTN`
	`A abcde`
	`C fghij`
	`G klmno`
	`T pqrst`
	`N uvwxy`
	The values `a...y` should be input as a quoted space separated string: "`a b c ... y`". Lower scores are better, so matches should be less than mismatches e.g. a,g,m,s = `-5` (match), mismatch = `6`.

`--affineOpen value`	Set the penalty for opening an affine alignment. (Default = `10`)
`--affineExtend a`	Change affine (extension) gap penalty. Lower value allows more gaps. (Default = `0`)

Options for Overlap/Dynamic Programming Alignments and Pairwise Overlap for *de novo* Assembly

Options	Description
`--useQuality`	Use substitution/insertion/deletion/merge quality values to score gap and mismatch penalties in pairwise alignments. As the insertion and deletion rates are much higher than substitution, this will make many alignments favor an insertion/deletion over a substitution. Naive consensus-calling methods will then often miss substitution polymorphisms. Use this option when calling consensus using the Quiver method. **Note**: When **not** using quality values to score alignments, there will be a lower consensus accuracy in homopolymer regions. (Default = `False`)
`--affineAlign`	Refine alignment using affine guided align. (Default = `False`)

Options for Filtering Reads

Options	Description
`--minReadLength l`	Skip reads that have a full length less than `l`. Subreads may be shorter. (Default = `50`)
`--minSubreadLength l`	Do **not** align subreads of length less than `l`. (Default = `0`)
`--maxScore m`	Maximum score to output; high is bad, negative is good. (Default = `0`)

Options for Parallel Alignment

Options	Description
`--nproc N`	Align using `N` processes. All large data structures such as the suffix array and tuple count table are shared. (Default = `1`)
`--start S`	Index of the first read to begin aligning. This is useful when multiple instances are running on the same data; for example when on a multi-rack cluster. (Default = `0`)
`--stride S`	Align one read every `S` reads. (Default = `1`)

Options for Subsampling Reads

Options	Description
`--subsample p`	Proportion `p` of reads to randomly subsample and align; expressed as a decimal. (Default = `0`)
`--help`	Displays help information and exits.
`--version`	Displays version information using the format `MajorVersion.Subversion.SHA1` (Example: `5.3.abcd123`) and exits.

Examples

To align reads from `reads.bam` to the ecoli_K12 genome, and output in PacBio BAM format:

```
blasr reads.bam ecoli_K12.fasta --bam --out ecoli_aligned.bam
```

To use multiple threads:

```
blasr reads.bam ecoli_K12.fasta --bam --out ecoli_aligned.bam --proc 16
```

To include a larger minimal match, for faster but less sensitive alignments:

```
blasr reads.bam ecoli_K12.fasta --bam --out ecoli_aligned.bam --proc 16 ––minMatch 15
```

To produce alignments in a pairwise human-readable format:

```
blasr reads.bam ecoli_K12.fasta -m 0
```

To use a precomputed suffix array for faster startup:

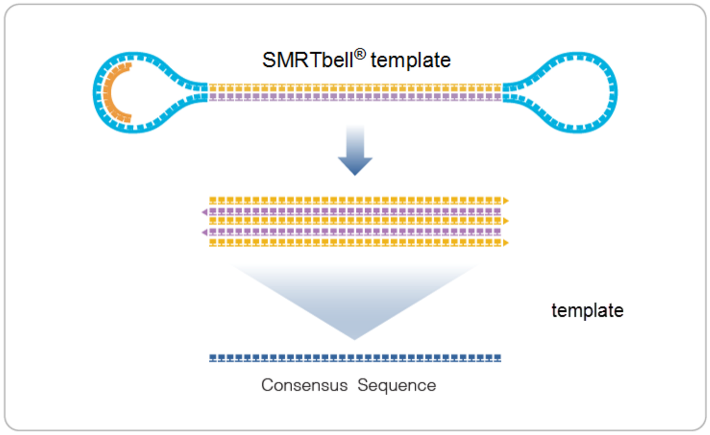
```
sawriter hg19.fasta.sa hg19.fasta #First precompute the suffix array
blasr reads.bam hg19.fasta --sa hg19.fasta.sa
```

To use a precomputed BWT-FM index for smaller runtime memory footprint, but slower alignments:

```
sa2bwt hg19.fasta hg19.fasta.sa hg19.fasta.bwt
blasr reads.bam hg19.fasta --bwt hg19.fasta.bwt
```

ccs Circular consensus sequencing (CCS) calculates consensus sequences from multiple "passes" around a circularized single DNA molecule (SMRTbell® template). CCS uses the Quiver framework to achieve optimal consensus results given the number of passes available.

Circular consensus sequencing (CCS)



Input Files

- One `.subreads.bam` file containing the subreads for each SMRTbell® template sequenced.

  Note: Sequence data generated by the PacBio RS II is in `bas.h5` format, while the sequence data generated by the Sequel System is in BAM file format. If you have a `bas.h5` file, you will need to convert it into a BAM file using the tool `bax2bam`, which simply needs the name of any `bas.h5` files to convert and the prefix of the output file. See "bax2bam" on page 5 for details.

Output Files

- A BAM file with one entry for each consensus sequence derived from a ZMW. BAM is a general file format for storing sequence data, which is described fully by the SAM/BAM working group. The CCS output format is a version of this general format, where the consensus sequence is represented by the "Query Sequence". Several tags were added to provide additional meta information. An example BAM entry for a consensus as seen by `samtools` is shown below.

```
m141008_060349_42194_c100704972550000001823137703241586_s1_p0/63/ccs   4     *     0     255
*       *       0       0       CCCGGGGATCCTCTAGAATGC    ~~~~~~~~~~~~~~~~~~~~~   RG:Z:83ba013f
np:i:35 rq:i:999        rs:B:i,37,0,0,1,0       sn:B:f,11.3175,6.64119,11.6261,14.5199
za:f:2.25461  zm:i:63 zs:B:f,-
1.57799,3.41424,2.96088,2.76274,3.65339,2.89414,2.446,3.04751,2.35529,3.65944,2.76774,4.119,1.679
81,1.66385,3.59421,2.32752,4.17803,-0.00353378,nan,0.531571,2.21918,3.88627,-
0.382997,0.650671,3.28113,0.798569,4.052,0.933297,3.00698,2.87132,2.66324,0.160431,1.99552,1.6935
4,1.90644,1.64448,3.13003,1.19977
```

Following are some of the common fields contained in the output BAM file:

Field	Description
`Query Name`	Movie Name / ZMW # /ccs
`FLAG`	Required by the format but meaningless in this context. Always set to 4 to indicate the read is unmapped.
`Reference Name`	Required by the format but meaningless in this context. Always set to `*`.
`Mapping Start`	Required by the format but meaningless in this context. Always set to `0`.
`Mapping Quality`	Required by the format but meaningless in this context. Always set to `255`.
`CIGAR`	Required by the format but meaningless in this context. Always set to `*`.
`RNEXT`	Required by the format but meaningless in this context. Always set to `*`.
`PNEXT`	Required by the format but meaningless in this context. Always set to `0`.
`TLEN`	Required by the format but meaningless in this context. Always set to `0`.
`Consensus Sequence`	The consensus sequence generated.
`Quality Values`	The per-base parametric quality metric. For details see the "Interpretting QUAL Values" section on page 15.
`RG Tag`	The read group identifier.
`bc Tag`	A 2-entry array of upstream-provided barcode calls for this ZMW.
`bq Tag`	The quality of the barcode call. (**Optional**: Depends on barcoded inputs.)
`np Tag`	The number of full passes that went into the subread. (**Optional**: Depends on barcoded inputs.)
`rq Tag`	The predicted read quality.
`rs Tag`	An array of counts for the effect of adding each subread. The first element indicates the number of success and the remaining indicate the number of failures. This is a comma-separated list of the number of reads successfully added, failed to converge in likelihood, failed the Z filtering, failed to pass the pre-POA size filtering, or were excluded for another reason.
`za Tag`	The average Z-score for all reads successfully added.
`zm Tag`	The ZMW hole number.
`zs Tag`	This is a comma-separated list of the Z-scores for each subread when compared to the initial candidate template. A `nan` value indicates that the subread was **not** added.

Usage

```
ccs [OPTIONS] OUTPUT FILES...
```

Example

```
ccs --minLength=100 myCCS.bam myData.subreads.bam
```

Required	Description
`Output File Name`	The name of the output BAM file; comes after all other options listed. (Example = `myResult.bam`)
`Input Files`	The name of one or more `subread.bam` files to be processed. This comes at the end of the `ccs` command. (Example = `myData.subreads.bam`)

Options	Description
`--version`	Prints the version number.
`--reportFile`	Contains a result tally of the outcomes for all ZMWs that were processed. If no file name is given, the report is output to the file `ccs_report.txt` In addition to the count of successfully-produced consensus sequences, this file lists how many ZMWs failed various data quality filters (SNR too low, not enough full passes, and so on) and is useful for diagnosing unexpected drops in yield.
`--minSnr`	Removes data that is likely to contain deletions. SNR is a measure of the strength of signal for all 4 channels (A, C, G, T) used to detect base pair incorporation. The SNR can vary depending on where in the ZMW a SMRTbell template stochastically lands when loading occurs. SMRTbell templates that land near the edge and away from the center of the ZMW have a less intense signal, and as a result can contain sequences with more "missed" base pairs. This value sets the threshold for minimum required SNR for any of the four channels. Data with SNR < 3.75 is typically considered lower quality. (Default = `3.75`)
`--minReadScore`	The minimum value for the predicted quality of any subread used for CCS. Note that this filters the input to CCS (the subread quality must be **above** this value), whereas the `--minPredictedAccuracy` option filters the output (the predicted consensus sequence must be **above** a certain predicted accuracy). (Default = `0.75`)
`--minLength`	The minimum length requirement for the median size of insert reads to generate a consensus sequence. If the targeted template is known to be a particular size range, this can filter out alternative DNA templates. (Default = `10`)
`--maxLength`	The maximum length for both subreads that will be processed as well as the consensus sequence that will be generated. For robust results while avoiding unnecessary computation on unusual data, set to ~20% above the largest expected insert size. (Default = `7000`)
`--minPasses`	The minimum number of passes for a ZMW to be emitted. This is the number of full passes. Full passes **must** have an adapter hit before and after the insert sequence and so do **not** include any partial passes at the start and end of the sequencing reaction. Additionally, the full pass count does **not** include any reads that were dropped by the Z-Filter. (Default = `3`)
`--minPredictedAccuracy`	The minimum predicted accuracy of a read. CCS generates an accuracy prediction for each read, defined as the expected percentage of matches in an alignment of the consensus sequence to the true read. A value of `0.99` indicates that only reads expected to be 99% accurate are emitted. (Default = `0.9`)
`--minZScore`	The minimum Z-Score for a subread to be included in the consensus generating process. For more information, see the "What are Z-Scores" section on page 15. (Default = `3.5`)
`--maxDropFraction`	The maximum number of subreads that can be dropped before the entire ZMW is discarded. Subreads that appear very unlikely given the initial template (low Z-score), are discarded **before** generating the consensus sequence as part of an initial quality filter. Typically, very few reads should be discarded but if a high proportion are, then the entire ZMW is dropped. (Default = `.34`)
`--zmws`	If the consensus sequence for only a subset of ZMWs is required, they can be specified here. ZMWs can be specified either by range (`--zmws=1-2000`) by values (`--zmws=5,10,20`), or by both (`--zmws=5-10,35,1000-2000`). Use a comma-separated list with no spaces.
`--numThreads`	How many threads to use while processing. By default, CCS will use as many threads as there are available cores to minimize processing time, but fewer threads can be specified here.
`--logFile`	The name of a log file to use. If none is given, the logging information is printed to `STDERR`. (Example: `mylog.txt`)
`--logLevel`	Specifies verbosity of log data to produce. By setting `--logLevel=DEBUG`, you can obtain detailed information on what ZMWs were dropped during processing, as well as any errors which may have appeared. (Default = `INFO`)

Options	Description
`--noPolish`	After constructing the initial template, do **not** proceed with the polishing steps. This is significantly faster, but generates less accurate data with no RQ or QUAL values associated with each base.
`--byStrand`	Separately generate a consensus sequence from the forward and reverse strands. Useful for identifying heteroduplexes formed during sample preparation.
`--force`	Overwrite the output file when you don't care that it already exists.

Interpreting QUAL Values

The QUAL value of a read is a measure of the posterior likelihood of an error at a particular position. Increasing QUAL values are associated with a decreasing probability of error. For indels and homopolymers, there is ambiguity as to which QUAL value is associated with the error probability. Shown below are different types of alignment errors, with a `*` indicating which sequence BP should be associated with the alignment error.

Mismatch

```
          *
ccs: ACGTATA
ref: ACATATA
```

Deletion

```
          *
ccs: AC-TATA
ref: ACATATA
```

Insertion

```
        *
ccs: ACGTATA
ref: AC-TATA
```

Homopolymer Insertion or Deletion

Indels should always be left-aligned, and the error probability is only given for the first base in a homopolymer.

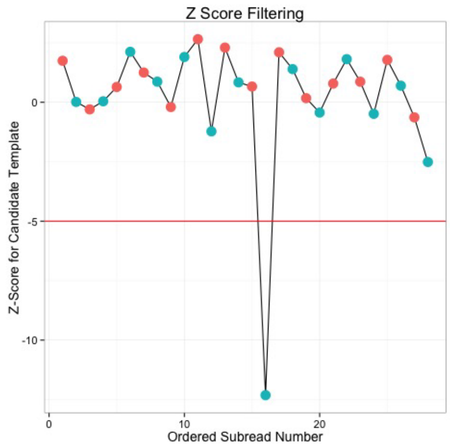
```
        *                     *
ccs: ACGGGGTATA     ccs: AC-GGGTATA
ref: AC-GGGTATA     ref: ACGGGGTATA
```

What are Z-Scores?

Z-score filtering is a way to remove outliers and contaminating data from the CCS data prior to consensus generation, a crucial step for any analysis.

The Z-score for a subread is a metric which quantifies how it fits the model or assumptions of CCS scoring. In CCS, an initial template sequence is proposed, and then further refined using data in the templates. The initial template is usually quite close to the final consensus sequence, and at this stage CCS will evaluate how likely each read is based on the candidate template. The likelihood of a read for a template is summarized by its Z-score, which asymptotically is normally distributed with a mean near 0.

Subreads with very low Z-scores are very **unlikely** to have been produced according to the CCS model, and so represent outliers. For example, the plot below shows the Z-scores for several subreads. With a -5 cutoff, we can see that one subread is excluded from the data.



CCS Yield Report

The CCS Report specifies the number of ZMWs that successfully produced consensus sequences, as well as a count of how many ZMWs did not produce a consensus sequence for various reasons. The entries in this report, as well as parameters used to increase of decrease the number of ZMWs that pass various filters, are explained in the table below.

ZMW Results	Parameters Affecting Results	Description
Below SNR threshold	`--minSnr`	ZMW had at least one channel's SNR below the minimum threshold.
No usable subreads	`--minReadScore`, `--minLength`, `--maxLength`	The ZMW had no usable subreads. Either there were no subreads, or all the subreads were below the minimum quality threshold, or were above/below the specified length thresholds.
Insert size too long	`--maxLength`	The consensus sequence was above the maximum length threshold. Note that if all the input subreads were already below this threshold, they would **all** have been excluded, leading to a "No usable subreads" result.
Insert size too small	`--minLength`	The consensus sequence was below the minimum length threshold. Note that if all the input subreads were already below this threshold, they would **all** have been excluded, leading to a "No usable subreads" result.
Not enough full passes	`--minPasses`	There were not enough subreads that had an adapter at the start and end of the subread (a "full pass").

ZMW Results	Parameters Affecting Results	Description
Too many unusable subreads	`--minZScore`, `--maxDropFraction`	The ZMW had too many subreads that could not be used. A read can be unusable if it appears too unlikely given the initial template (low Z-score), or rarely, if a numerical rounding error occurs during processing.
CCS did not converge	None	The consensus sequence did not converge after the maximum number of allowed rounds of polishing.
CCS below minimum predicted accuracy	`--minPredictedAccuracy`	Each CCS read has a predicted level of accuracy associated with it. Reads that are below the minimum specified threshold are removed.
Unknown error during processing	None	These should not occur.

dataset

The `dataset` tool creates, opens, manipulates and writes Data Set XML files. The commands allow you to perform operations on the various types of data held by a Data Set XML: merge, split, write, and so on.

Usage

```
dataset [-h] [--version] [--log-file LOG_FILE]
        [--log-level {DEBUG,INFO,WARNING,ERROR,CRITICAL} | --debug | --quiet | -v]
        [--strict] [--skipCounts]

{create,filter,merge,split,validate,summarize,consolidate,loadstats,newuuid,loadmetada
ta,copyto,absolutize,relativize}
```

Options	Description
`-h, --help`	Displays help information and exits.
`<Command> -h`	Displays help for a specific command.
`-v, --version`	Displays program version number and exits.
`--log-file LOG_FILE`	Write the log to file. (Default = `None`, writes to `stdout`.)
`--log-level`	Specify the log level; values are `[DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL]`. (Default = `INFO`)
`--debug`	Alias for setting the log level to `DEBUG`. (Default = `False`)
`--quiet`	Alias for setting the log level to `CRITICAL` to suppress output. (Default = `False`)
`-v`	Sets the verbosity level. (Default = `NONE`)
`--strict`	Turn on strict tests and display all errors. (Default = `False`)
`--skipCounts`	Skip updating `NumRecords` and `TotalLength` counts. (Default = `False`)

`create` Command: Create an XML file from a `fofn` (file-of-file names) or BAM file. Possible types: `SubreadSet, AlignmentSet, ReferenceSet, HdfSubreadSet, BarcodeSet, ConsensusAlignmentSet, ConsensusReadSet, ContigSet`.

```
dataset create [-h] [--type DSTYPE] [--name DSNAME] [--generateIndices]
               [--metadata METADATA] [--novalidate] [--relative]
               outfile infile [infile ...]
```

Required	Description
`outfile`	The name of the XML file to create.
`infile`	The `fofn` (file-of-file-names) or BAM file(s) to convert into an XML file.

Options	Description
`--type DSTYPE`	Specify the type of XML file to create. (Default = `NONE`)
`--name DSNAME`	The name of the new Data Set XML file.
`--generateIndices`	Generate index files (`.pbi` and `.bai` for BAM, `.fai` for FASTA). Requires `samtools`/`pysam` and `pbindex`. (Default = `FALSE`)
`--metadata METADATA`	A `metadata.xml` file (or Data Set XML) to supply metadata. (Default = `NONE`)
`--novalidate`	Don't validate the resulting XML. Leaves the paths as they are.
`--relative`	Make the included paths relative instead of absolute. This is **not** compatible with `--novalidate`.

`filter` Command: Filter an XML file using filters and threshold values.

- Suggested filters: [`accuracy`, `bc`, `bcf`, `bcq`, `bcr`, `bq`, `cx`, `length`, `movie`, `n_subreads`, `pos`, `qend`, `qname`, `qstart`, `readstart`, `rname`, `rq`, `tend`, `tstart`, `zm`].
- More resource-intensive filter: [`qs`]

Note: Multiple filters with different names will be ANDed together. Multiple filters with the **same** name will be ORed together, duplicating existing requirements.

```
dataset filter [-h] infile outfile filters [filters ...]
```

Required	Description
`infile`	The name of the XML file to filter.
`outfile`	The name of the output filtered XML file.
`filters`	The values to filter on. (Example: `rq>0.85`)

`merge` Command: Combine XML files.

```
dataset merge [-h] outfile infiles [infiles ...]
```

Required	Description
`infiles`	The names of the XML files to merge.
`outfile`	The name of the output XML file.

`split` Command: Split a Data Set XML file.

```
dataset split [-h] [--contigs] [--barcodes] [--zmws] [--byRefLength]
                   [--noCounts] [--chunks CHUNKS] [--maxChunks MAXCHUNKS]
```

```
[--targetSize TARGETSIZE] [--breakContigs]
[--subdatasets] [--outdir
infile [outfiles...]
```

Required	Description
`infile`	The name of the XML file to split.

Options	Description
`outfiles`	The names of the resulting XML files.
`--contigs`	Split the XML file based on contigs. (Default = `FALSE`)
`--barcodes`	Split the XML file based on barcodes. (Default = `FALSE`)
`--zmws`	Split the XML file based on ZMWs. (Default = `FALSE`)
`--byRefLength`	Split contigs by contig length. (Default = `TRUE`)
`--noCounts`	Update the Data Set counts after the split. (Default = `FALSE`)
`--chunks x`	Split contigs into `x` total windows. (Default = `0`)
`--maxChunks x`	Split contig list into at most `x` groups. (Default = `0`)
`--targetSize x`	The minimum number of records per chunk. (Default = `5000`)
`--breakContigs`	Break contigs to get closer to `maxCounts`. (Default = `False`)
`--subdatasets`	Split the XML file based on subdatasets. (Default = `False`)
`--outdir OUTDIR`	Specify an output directory for the resulting XML files. (Default = `<in-place>`, **not** the current working directory.)

`validate` Command: Validate XML and ResourceId files. (This is an internal testing functionality that may be useful.)

Note: This command requires that `pyxb` (**not** distributed with SMRT Link) be installed. If **not** installed, `validate` simply checks that the files pointed to in ResourceIds exist.

```
dataset validate [-h] [--skipFiles] infile
```

Required	Description
`infile`	The name of the XML file to validate.

Options	Description
`--skipFiles`	Skip validating external resources. (Default = `False`)

`summarize` Command: Summarize a Data Set XML file.

```
dataset summarize [-h] infile
```

Required	Description
`infile`	The name of the XML file to summarize.

`consolidate` Command: Consolidate XML files.

```
dataset consolidate [-h] [--numFiles NUMFILES] [--noTmp]
                    infile datafile xmlfile
```

Required	Description
`infile`	The name of the XML file to consolidate.
`datafile`	The name of the resulting data file.
`xmlfile`	The name of the resulting XML file.

Options	Description
`--numFiles x`	The number of data files to produce. (Default = `1`)
`--noTmp`	Do **not** copy to a temporary location to ensure local disk use. (Default = `False`)

`loadstats` Command: Load an `sts.xml` file containing pipeline statistics into a Data Set XML file.

```
dataset loadstats [-h] [--outfile OUTFILE] infile statsfile
```

Required	Description
`infile`	The name of the Data Set XML file to modify.
`statsfile`	The name of the `.sts.xml` file to load.

Options	Description
`--outfile OUTFILE`	The name of the XML file to output. (Default = `None`)

`newuuid` Command: Refresh a Data Set's Unique ID.

```
dataset newuuid [-h] [--random] infile
```

Required	Description
`infile`	The name of the XML file to refresh.

Options	Description
`--random`	Generate a random UUID, instead of a hash. (Default = `False`)

`loadmetadata` Command: Load a `.metadata.xml` file into a Data Set XML file.

```
dataset loadmetadata [-h] [--outfile OUTFILE] infile metadata
```

Required	Description
`infile`	The name of the Data Set XML file to modify.
`metadata`	The `.metadata.xml` file to load, or Data Set to borrow from.

Options	Description
`--outfile OUTFILE`	The XML file to output. (Default = `None`)

`copyto` Command: Copy a Data Set and resources to a new location.

```
dataset copyto [-h] [--relative] infile outdir
```

Required	Description
`infile`	The name of the XML file to copy.
`outdir`	The directory to copy to.

Options	Description
`--relative`	Make the included paths relative instead of absolute. (Default = `False`)

`absolutize` Command: Make the paths in an XML file absolute.

```
dataset absolutize [-h] [--outdir OUTDIR] infile
```

Required	Description
`infile`	The name of the XML file whose paths should be absolute.

Options	Description
`--outdir OUTDIR`	Specify an optional output directory. (Default = `None`)

`relativize` Command: Make the paths in an XML file relative.

```
dataset relativize [-h] infile
```

Required	Description
`infile`	The name of the XML file whose paths should be relative.

Example - Filter Reads

To filter one or more BAM file's worth of subreads, aligned or otherwise, and then place them into a single BAM file:

```
# usage: dataset filter <in_fn.xml> <out_fn.xml> <filters>
dataset filter in_fn.subreadset.xml filtered_fn.subreadset.xml 'rq>0.85'

# usage: dataset consolidate <in_fn.xml> <out_data_fn.bam> <out_fn.xml>
dataset consolidate filtered_fn.subreadset.xml consolidate.subreads.bam
out_fn.subreadset.xml
```

The filtered Data Set and the consolidated Data Set should be read-for-read equivalent when used with SMRT Analysis software.

Example - Resequencing Pipeline

- Align two movie's worth of subreads in two SubreadSets to a reference.
- Merge the subreads together.
- Split the subreads into Data Set chunks by contig.
- Process using `quiver` on a chunkwise basis (in parallel).

1. Align each movie to the reference, producing a Data Set with one BAM file for each execution:

```
pbalign movie1.subreadset.xml referenceset.xml movie1.alignmentset.xml
pbalign movie2.subreadset.xml referenceset.xml movie2.alignmentset.xml
```

2. Merge the files into a FOFN-like Data Set; BAMs are **not** touched:

```
# dataset merge <out_fn> <in_fn> [<in_fn> <in_fn> ...]
dataset merge merged.alignmentset.xml movie1.alignmentset.xml movie2.alignmentset.xml
```

3. Split the Data Set into chunks by contig name; BAMs are **not** touched:
   - Note that supplying output files splits the Data Set into that many output files (up to the number of contigs), with multiple contigs per file.
   - **Not** supplying output files splits the Data Set into **one** output file per contig, named automatically.
   - Specifying a number of chunks instead will produce that many files, with contig or even subcontig (reference window) splitting.

```
dataset split --contigs --chunks 8 merged.alignmentset.xml
```

4. Process the chunks using `Quiver`:

```
variantCaller --alignmentSetRefWindows --referenceFileName referenceset.xml --
outputFilename chunk1consensus.fasta --algorithm quiver chunk1contigs.alignmentset.xml

variantCaller --alignmentSetRefWindows --referenceFileName referenceset.xml --
outputFilename chunk2consensus.fasta --algorithm quiver chunk2contigs.alignmentset.xml
```

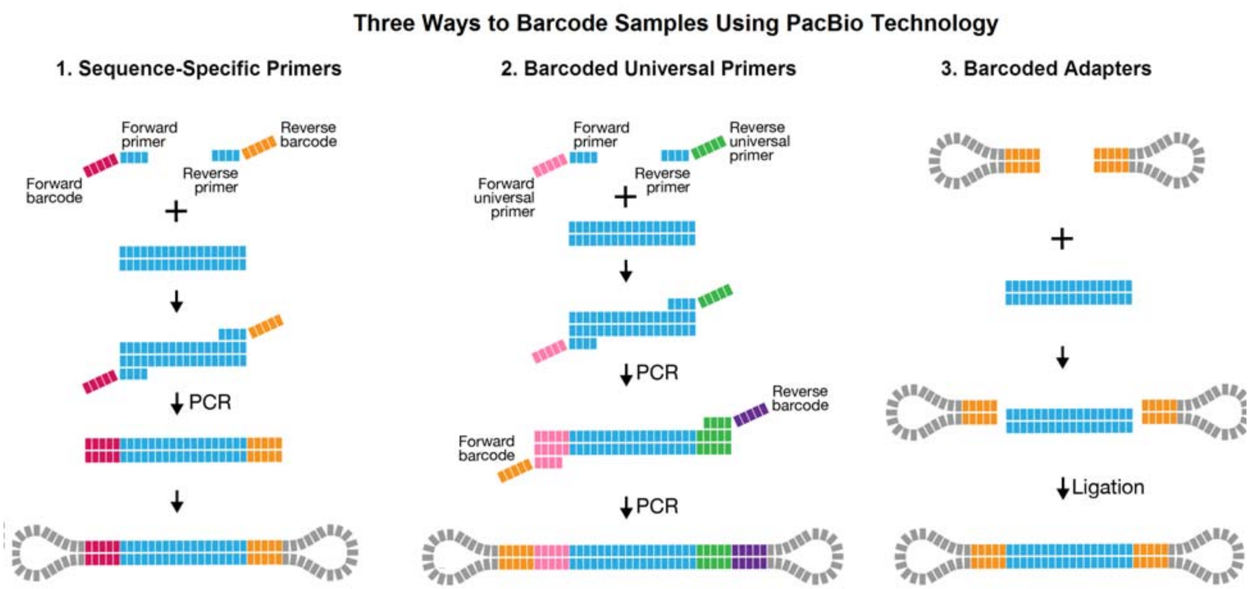
The chunking works by duplicating the original merged Data Set (no BAM duplication) and adding filters to each duplicate such that only reads belonging to the appropriate contigs are emitted. The contigs are distributed among the output files in such a way that the total number of records per chunk is about even.

Demultiplex Barcodes

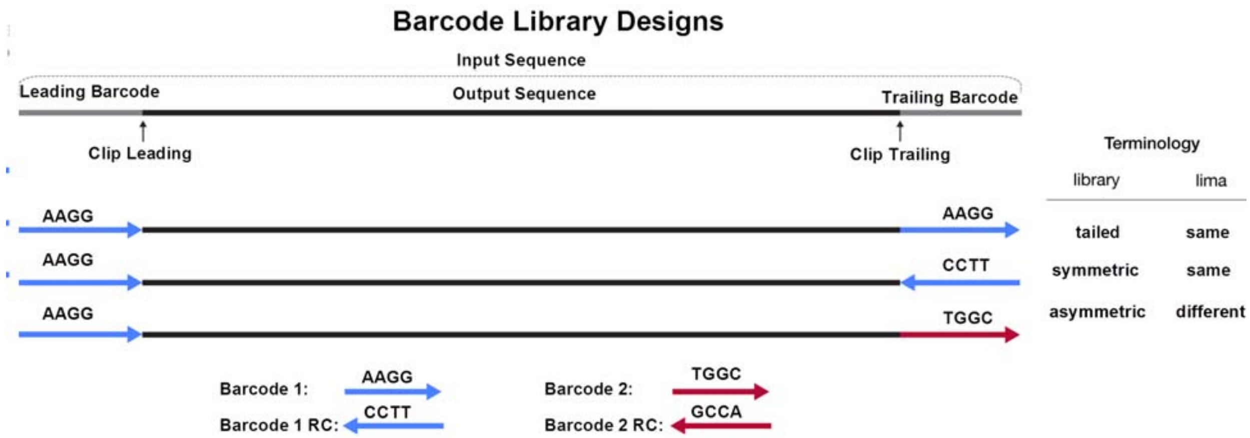
The **Demultiplex Barcodes** application identifies barcode sequences in PacBio single-molecule sequencing data. It **replaces** `pbbarcode` and `bam2bam` for demultiplexing, starting with SMRT Analysis v5.1.0. The core alignment algorithm in the new Demultiplex Barcodes application is the same, but the algorithm to identify barcode pairs, as well as usability, are improved.

Demultiplex Barcodes can demultiplex samples that have a unique per-sample barcode pair and were pooled and sequenced on the same SMRT Cell. There are three different methods for barcoding samples with PacBio technology:

1. Sequence-specific primers
2. Barcoded universal primers
3. Barcoded adapters



In addition, there are three different barcode library designs. As **Demultiplex Barcodes** supports raw subread and ccs read demultiplexing, the following terminology is based on the per (sub-) read view.



In the overview above, the input sequence is flanked by adapters on both sides. The bases adjacent to an adapter are **barcode regions**. A read can have up to two barcode regions, leading and trailing. Either or both adapters can be missing and consequently the leading and/or trailing region is not being identified.

For **symmetric** and **tailed** library designs, the **same** barcode is attached to both sides of the insert sequence of interest. The only difference is the orientation of the trailing barcode. For barcode identification, one read with a single barcode region is sufficient.

For the **asymmetric** design, **different** barcodes are attached to the sides of the insert sequence of interest. To identify the different barcodes, a read with leading and trailing barcode regions is required.

Output barcode pairs are generated from the identified barcodes. The barcode names are combined using “`--`“, for example `bc1002--bc1054`. The sort order is defined by the barcode indices, starting with the lowest.

Workflow

By default, **Demultiplex Barcodes** processes input reads grouped by ZMW, **except** if the `--per-read` option is used. All barcode regions along the read are processed individually. The final per-ZMW result is a summary over all barcode regions. Each ZMW is assigned to a pair of selected barcodes from the provided set of candidate barcodes. Subreads from the same ZMW will have the same barcode and barcode quality. For a particular target barcode region, every barcode sequence gets aligned as given and as reverse-complement, and higher scoring orientation is chosen. This results in a list of scores over all candidate barcodes.

- If only **same** barcode pairs are of interest (symmetric/tailed), use the `--same` option to filter out **different** barcode pairs.
- If only **different** barcode pairs are of interest (asymmetric), use the `--min-passes 1` option to require at least two barcodes to be read.

Half Adapters

For an adapter call with only one barcode region, the high-quality region finder cuts right through the adapter. The preceding or succeeding subread was too short and was removed, or the sequencing reaction started/stopped there. This is called a **half adapter**. Thus, there are also 1.5, 2.5, N+0.5 adapter calls.

ZMWs with half or only one adapter can be used to identify same barcode pairs; positive-predictive value might be reduced compared to high adapter calls. For asymmetric designs with different barcodes in a pair, at least a single full-pass read is required; this can be two adapters, two half adapters, or a combination.

Usage:

Note: Any existing output files will be **overwritten** after execution.

Analysis of subread data:

```
lima movie.subreads.bam barcodes.fasta prefix.bam
lima movie.subreadset.xml barcodes.barcodeset.xml prefix.subreadset.xml
```

Analysis of CCS data:

```
lima --css movie.ccs.bam barcodes.fasta prefix.bam
lima --ccs movie.consensusreadset.xml barcodes.barcodeset.xml
prefix.consensusreadset.xml
```

If you do not need to import the demultiplexed data into SMRT Link, use the `--no-pbi` option to minimize memory consumption and run time.

Asymmetric options:

```
Raw: --min-passes 1
CCS: --ccs
```

Symmetric or Tailed options:

```
Raw: --same
CCS: --same --ccs
```

Options	Description
`--same`	Retain only reads with the same barcodes on both ends of the insert sequence, such as symmetric and tailed designs.
`--min-length n`	Reads with lengths below n base pairs after demultiplexing are omitted. ZMWs with no reads passing are omitted. (Default = `50`)
`--max-input-length n`	Reads with lengths above n base pairs are omitted for scoring in the demultiplexing step. (Default = `0`, deactivated)
`--min-score n`	ZMWs with barcode scores below n are omitted. A **barcode score** measures the alignment between a barcode attached to a read and an ideal barcode sequence, and is an indicator how well the chosen barcode pair matches. It is normalized to a range between `0` (no hit) and `100` (a perfect match). (Default = `0,` Pacific Biosciences recommends setting it to `26`.)

Options	Description
--min-passes n	ZMWs with less than n full passes, a read with a leading and trailing adapter, are omitted. (Default = 0, no full-pass needed) Example: 0 pass : insert - adapter - insert 1 pass : insert - adapter - INSERT - adapter - insert 2 passes: insert - adapter - INSERT - adapter - INSERT - adapter - insert
--score-full-pass	Only use reads flanked by adapters on both sides, full-pass reads, for barcode identification.
--per-read	Score and tag per subread, instead of per ZMW.
--ccs	Set defaults to -A 1 -B 4 -D 3 -I 3 -X 1.
--peek n	Looks at the first n ZMWs of the input and return the mean. This lets you test multiple test barcode.fasta files and see which set of barcodes was used.
--guess n	This performs demultiplexing twice. In the first iteration, **all** barcodes are tested per ZMW. Afterwards, the barcode occurrences are counted and their mean is tested against the threshold n; only those barcode pairs that pass this threshold are used in the second iteration to produce the final demultiplexed output. A prefix.lima.guess file shows the decision process; --same is being respected.
--guess-min-count	The minimum ZMW count to whitelist a barcode. This filter is ANDed with the minimum barcode score specified by --guess. (Default = 0)
--peek && --guess	The optimal way is to use both advanced options in combination, such as, --peek 1000 --guess 45. lima will run twice on the input data. For the first 1000 ZMWs, lima will guess the barcodes and store the mask of identified barcodes. In the second run, the barcode mask is used to demultiplex all ZMWs.
--single-side	Identify barcodes in molecules that only have barcodes adjacent to one adapter.
--var-barcode-length	Allow barcodes to have different lengths. This will increase run time.
--window-size-mult --window-size-bp	The candidate region size multiplier: barcode_length * multiplier. (Default = 1.5) Optionally, you can specify the region size in base pairs using --window-size-bp.
--num-threads n	Spawn n threads; 0 means use all available cores. This option also controls the number of threads used for BAM and PBI compression. (Default = 0)
--chunk-size n	Each thread consumes n ZMWs per chunk for processing. (Default = 10).
--no-bam	Do **not** produce BAM output. Useful if only reports are of interest, as run time is shorter.
--no-pbi	Do **not** produce a .bam.pbi index file. The on-the-fly .bam.pbi file generation buffers the output data. If you do not need a .bam.pbi index file for SMRT Link import, use this option to decrease memory usage to a minimum and shorten the run time.
--no-reports	Do **not** produce any reports. Useful if only demultiplexed BAM files are needed.
--dump-clips	Output all clipped barcode regions generated to the <prefix>.lima.clips file.
--dump-removed	Output all records that did **not** pass the specified thresholds, or are without barcodes, to the <prefix>.lima.removed.bam file.

Options	Description
`--split-bam` `--split-bam-named`	Each barcode has its own BAM file called `prefix.idxBest-idxCombined.bam`, such as `prefix.0-0.bam`. Optionally ,`--split-bam-named` names the files by their barcode names instead of their barcode indices.

Input Files:

Input data in PacBio-enhanced BAM format is either:

- Sequence data **-** Unaligned subreads, directly from a Sequel instrument, or
- Unaligned ccs reads, generated by CCS 2.

Note: To demultiplex PacBio RS II data, use SMRT Link or `bax2bam` to convert `.h5` files to BAM format.

Barcodes are provided as a FASTA file:

- One entry per barcode sequence.
- **No** duplicate sequences.
- All bases must be in **upper-case**.
- Orientation-agnostic (forward or reverse-complement, but **not** reversed).

Example:

```
>bc1000
CTCTACTTACTTACTG
>bc1001
GTCGTATCATCATGTA
>bc1002
AATATACCTATCATTA
```

Note: Name barcodes using an alphabetic character prefix to avoid later barcode name/index confusion.

Output Files:

Demultiplex Barcodes generates multiple output files by default, all starting with the same prefix as the output file, using the suffixes `.bam`, `.subreadset.xml`, and `.consensusreadset.xml`. The report prefix is `lima`. Example:

```
lima m54007_170702_064558.subreads.bam barcode.fasta /my/path/
m54007_170702_064558_demux.subreadset.xml
```

For all output files, the prefix will be `/my/path/m54007_170702_064558_demux`.

- `<prefix>.bam`: Contains clipped records, annotated with barcode tags, that passed filters and respect the `--same` option.
- `<prefix>.lima.report`: A tab-separated file describing each ZMW, unfiltered. This is useful information for investigating the demultiplexing process and the underlying data. A single row contains **all** reads from a single ZMW. For `--per-read`, each row contains one subread, and ZMWs might span multiple rows.
- `<prefix>.lima.summary`: Lists how many ZMWs were filtered, how many ZMWs are the same or different, and how many reads were filtered.

```
(1)
ZMWs input                (A) : 213120
ZMWs above all thresholds (B) : 176356 (83%)
ZMWs below any threshold  (C) : 36764 (17%)

(2)
ZMW marginals for (C):
Below min length              : 26 (0%)
Below min score               : 0 (0%)
Below min passes              : 0 (0%)
Below min score lead          : 11656 (32%)
Without adapter               : 25094 (68%)

(3)
ZMWs for (B):
With same barcode             : 162244 (92%)
With different barcodes       : 14112 (8%)
Coefficient of correlation    : 32.79%

(4)
ZMWs for (A):
Allow diff barcode pair       : 157264 (74%)
Allow same barcode pair       : 188026 (88%)

(5)
Reads for (B):
Above length                  : 1278461 (100%)
Below length                  : 2787 (0%)
```

Explanation of each block:

1. Number of ZMWs that went into `lima`, how many ZMWs were passed to the output file, and how many did not qualify.
2. For those ZMWs that did not qualify: The marginal counts of each filter. (Filter are described in the **Options** table.)
3. For those ZMWs that passed: How many were flagged as having the same or different barcode pair, as well as the coefficient of variation for the barcode ZMW yield distribution in percent.
4. For all input ZMWs: How many allow calling the same or different barcode pair. This is a simplified version of how many ZMW have at least one full pass to allow a different barcode pair call and how many ZMWs have at least half an adapter, allowing the same barcode pair call.
5. For those ZMWs that qualified: The number of reads that are above and below the specified `--min-length` threshold.

- `<prefix>.lima.counts`: A .tsv file listing the counts of each observed barcode pair. Only passing ZMWs are counted. Example:

```
$ column -t prefix.lima.counts
IdxFirst  IdxCombined      IdxFirstNamed  IdxCombinedNamed
Counts
1         1                bc1002         bc1002            113
14        14               bc1015         bc1015            129
18        18               bc1019         bc1019            106
```

- `<prefix>.lima.clips`: Contains clipped barcode regions generated using the `--dump-clips` option. Example:

```
$ head -n 6 prefix.lima.clips
>m54007_170702_064558/4850602/6488_6512 bq:34 bc:11
CATGTCCCCTCAGTTAAGTTACAA
>m54007_170702_064558/4850602/6582_6605 bq:37 bc:11
TTTTGACTAACTGATACCAATAG
>m54007_170702_064558/4916040/4801_4816 bq:93 bc:10
```

- `<prefix>.lima.removed.bam`: Contains records that did **not** pass the specified thresholds, or are without barcodes, using the option `--dump-removed`.
  `lima` does **not** generate a .pbi, nor Data Set for this file. This option **cannot** be used with any splitting option.
- `<prefix>.lima.guess`: A .tsv file that describes the barcode subsetting process activated using `--peek` and `--guess` options.

IdxFirst	IdxCombined	IdxFirstNamed	IdxCombinedNamed	NumZMWs	MeanScore	Picked
0	0	bc1001t	bc1001t	1008	50	1
1	1	bc1002t	bc1002t	1005	60	1
2	2	bc1003t	bc1003t	5	24	0
3	3	bc1004t	bc1004t	555	61	1

- One DataSet,`.subreadset.xml`, or `.consensusreadset.xml` file is generated per output BAM file.
- `.pbi`: One PBI file is generated per output BAM file.

fasta-to-gmap-reference

The `fasta-to-gmap-reference` tool converts a reference FASTA file to a GMAP database (including the index files required by GMAP), and creates a GmapReferenceSet XML file. The GmapReferenceSet XML file can be imported into SMRT Link and used as a reference with the **Iso-Seq® with Mapping** application. (For SMRT Link v5.1.0, this is a **mandatory** step for using the application, as SMRT Link cannot generate this Data Set type itself.)

Usage

```
fasta-to-gmap-reference [options] fasta-file output-dir name
```

Required	Description
`fasta-file`	The path to the reference FASTA file.
`output-dir`	The location for the output GMAP database and Data Set XML file.

Required	Description
name	The name of the output GmapReferenceSet XML file.

Options	Description
--organism <value>	The name of the organism.
--ploidy <value>	Ploidy.
--in-place	Do **not** copy the input FASTA file to the output location.
--log2stdout	If true, log output will be displayed to the console. (Default = False)
--log-level <value>	Specify the log level; values are [ERROR, WARN, DEBUG, INFO.] (Default = ERROR)
--debug	Same as --log-level DEBUG.
--quiet	Same as --log-level ERROR.
--verbose	Same as --log-level INFO.
--log-file <value>	Log output file name. (Default = ".")
--logback <value>	Override all logger configuration with the specified logback.xml file.

fasta-to-reference

The fasta-to-reference tool converts a FASTA file to a ReferenceSet Data Set XML that contains the required index files:

- samtools index (.fai)
- sawriter index (fasta.sa)
- SMRT View indexes (fasta.config.index and fasta.index)

fasta-to-reference is provided with SMRT Link, and requires the samtools, sawriter and ngmlr executables.

Note that fasta-to-reference will run on a single CPU on the host which it is executed, and not distributed on the cluster. For human-scale references, this may take up to half a day or more to run, and consumes a significant amount of memory. The indexing step with sawriter can use over 34 GB of memory. When running this program, make sure the process has sufficient compute resources and will not be interrupted. PacBio suggest redirecting stderr/stdout to a log file. For example:

```
fasta-to-reference hg38.fasta /opt/smrtlink/references hg38 --organism Homo_sapiens >
fasta2ref.log 2>&1
```

Usage

```
fasta-to-reference [options] fasta-file output-dir name
```

Required	Description
fasta-file	The path to the input FASTA file.
output-dir	The path to the output PacBio Reference Dataset XML.

Required	Description
name	The name of the ReferenceSet.

Options	Description
--organism <value>	The name of the organism.
--ploidy <value>	Ploidy.
-d, --debug	Specifies logging to stdout.
-h, --help	Displays help information and exits.

Input File

- *.fasta file to convert.

Output Files

- *.referenceset.xml.
- fasta-enc.2.ngm and fasta-ht-13-2.2.ngm ngmlr indexes.

ipdSummary

The ipdSummary tool detects DNA base-modifications from kinetic signatures. It is part of the kineticsTool package.

kineticsTool loads IPDs observed at each position in the genome, compares those IPDs to value expected for unmodified DNA, and outputs the result of this statistical test. The expected IPD value for unmodified DNA can come from either an in-silico control or an amplified control. The in-silico control is trained by Pacific Biosciences and shipped with the package. It predicts the IPD using the local sequence context around the current position. An amplified control Data Set is generated by sequencing unmodified DNA with the same sequence as the test sample. An amplified control sample is usually generated by whole-genome amplification of the original sample.

Modification Detection

The basic mode of kineticsTool does an independent comparison of IPDs at each position on the genome, for each strand, and outputs various statistics to CSV and GFF files (after applying a significance filter.)

Modifications Identification

kineticsTool also has a Modification Identification mode that can decode multi-site IPD 'fingerprints' into a reduced set of calls of specific modifications. This feature has the following benefits:

- Different modifications occuring on the same base can be distinguished; for example, m5C and m4C.
- The signal from one modification is combined into one statistic, improving sensitivity, removing extra peaks, and correctly centering the call.

Algorithm: Synthetic Control

Studies of the relationship between IPD and sequence context reveal that most of the variation in mean IPD across a genome can be predicted from a 12-base sequence context surrounding the active site of the DNA polymerase. The bounds of the relevant context window correspond to the window of DNA in contact with the polymerase, as seen in DNA/ polymerase crystal structures. To simplify the process of finding DNA modifications with PacBio data, the tool includes a pre-trained lookup table mapping 12-mer DNA sequences to mean IPDs observed in C2 chemistry.

Algorithm: Filtering and Trimming

`kineticsTool` uses the Mapping QV generated by `blasr` and stored in the `cmp.h5` or BAM file (or AlignmentSet) to ignore reads that are not confidently mapped. The default minimum Mapping QV required is 10, implying that `blasr` has 90% confidence that the read is correctly mapped. Because of the range of read lengths inherent in PacBio data, this can be changed using the `--mapQvThreshold` option.

There are a few features of PacBio data that require special attention to achieve good modification detection performance. `kineticsTool` inspects the alignment between the observed bases and the reference sequence for an IPD measurement to be included in the analysis. The PacBio read sequence **must** match the reference sequence for `k` around the cognate base. In the current module, `k=1`. The IPD distribution at some locus can be thought of as a mixture between the 'normal' incorporation process IPD, which is sensitive to the local sequence context and DNA modifications, and a contaminating 'pause' process IPD, which has a much longer duration (mean > 10 times longer than normal), but happen rarely (~1% of IPDs).
Note: Our current understanding is that pauses do **not** carry useful information about the methylation state of the DNA, however a more careful analysis may be warranted. Also note that modifications that drastically increase the roughly 1% of observed IPDs are generated by pause events. Capping observed IPDs at the global 99^{th} percentile is motivated by theory from robust hypothesis testing. Some sequence contexts may have naturally longer IPDs; to avoid capping too much data at those contexts, the cap threshold is adjusted per context as follows:

```
capThreshold = max(global99, 5*modelPrediction,
percentile(ipdObservations, 75))
```

Agorithm: Statistical Testing

We test the hypothesis that IPDs observed at a particular locus in the sample have longer means than IPDs observed at the same locus in unmodified DNA. If we have generated a Whole Genome Amplified Data Set, which removes DNA modifications, we use a case-control, two-sample t-test. This tool also provides a pre-calibrated 'synthetic control' model which predicts the unmodified IPD, given a 12-base sequence context. In the synthetic control case we use a one-sample t-test, with an adjustment to account for error in the synthetic control model.

Usage

To run using a BAM input, and output GFF and HDF5 files:

```
ipdSummary aligned.bam --reference ref.fasta  m6A,m4C --gff basemods.gff \
--csv_h5 kinetics.h5
```

To run using `cmp.h5` input, perform methyl fraction calculation, and output GFF and CSV files:

```
ipdSummary aligned.cmp.h5 --reference ref.fasta  m6A,m4C --methylFraction \
--gff basemods.gff --csv kinetics.csv
```

Output Options	Description
`--gff FILENAME`	GFF format.
`--csv FILENAME`	Comma-separated value format.
`--csv_h5 FILENAME`	Compact binary-equivalent of .csv, in HDF5 format.
`--bigwig FILENAME`	BigWig file format; mostly only useful for SMRT View.

Input Files

- A standard PacBio alignment file - either AlignmentSet XML, BAM, or `cmp.h5` - containing alignments and IPD information.
- Reference Sequence used to perform alignments. This can be either a FASTA file or a ReferenceSet XML.

Output Files

The tool provides results in a variety of formats suitable for in-depth statistical analysis, quick reference, and consumption by visualization tools such as SMRT View. Results are generally indexed by reference position and reference strand. In all cases the strand value refers to the strand carrying the modification in the DNA sample. Remember that the kinetic effect of the modification is observed in read sequences aligning to the opposite strand. So reads aligning to the positive strand carry information about modification on the negative strand and vice versa, but the strand containing the putative modification is always reported.

- `modifications.gff`: Compliant with the GFF Version 3 specification (**http://www.sequenceontology.org/gff3.shtml**). Each template position/strand pair whose probability value exceeds the probability value threshold appears as a row. The template position is 1-based, per the GFF specifications. The strand column refers to the strand carrying the detected modification, which is the opposite strand from those used to detect the modification. The GFF confidence column is a Phred-transformed probability value of detection.

The auxiliary data column of the GFF file contains other statistics which may be useful for downstream analysis or filtering. These include the coverage level of the reads used to make the call, and +/- 20 bp sequence context surrounding the site.

- `modifications.csv`: Contains one row for each (reference position, strand) pair that appeared in the Data Set with coverage at least `x`. `x` defaults to `3`, but is configurable with the `--minCoverage` option. The reference position index is 1-based for compatibility with the GFF file in the R environment. Note that this output type scales poorly and is **not** recommended for large genomes; the HDF5 output should perform much better in these cases.

Output Columns: In-Silico Control Mode

Column	Description
`refId`	Reference sequence ID of this observation.
`tpl`	1-based template position.
`strand`	Native sample strand where kinetics were generated. `0` is the strand of the original FASTA, `1` is opposite strand from FASTA.
`base`	The cognate base at this position in the reference.
`score`	Phred-transformed probability value that a kinetic deviation exists at this position.
`tMean`	Capped mean of normalized IPDs observed at this position.
`tErr`	Capped standard error of normalized IPDs observed at this position (`standard deviation/sqrt(coverage)`).
`modelPrediction`	Normalized mean IPD predicted by the synthetic control model for this sequence context.
`ipdRatio`	`tMean/modelPrediction`.
`coverage`	Count of valid IPDs at this position.
`frac`	Estimate of the fraction of molecules that carry the modification.
`fracLow`	2.5% confidence bound of the `frac` estimate.
`fracUpp`	97.5% confidence bound of the `frac` estimate.

Output Columns: Case Control Mode

Column	Description
`refId`	Reference sequence ID of this observation.
`tpl`	1-based template position.
`strand`	Native sample strand where kinetics were generated. `0` is the strand of the original FASTA, `1` is opposite strand from FASTA.
`base`	The cognate base at this position in the reference.
`score`	Phred-transformed probability value that a kinetic deviation exists at this position.
`caseMean`	Mean of normalized case IPDs observed at this position.
`controlMean`	Mean of normalized control IPDs observed at this position.
`caseStd`	Standard deviation of case IPDs observed at this position.

Column	Description
controlStd	Standard deviation of control IPDs observed at this position.
ipdRatio	tMean/modelPrediction.
testStatistic	T-test statistic.
coverage	Mean of case and control coverage.
controlCoverage	Count of valid control IPDs at this position.
caseCoverage	Count of valid case IPDs at this position.

juliet

`juliet` is a general-purpose minor variant caller that identifies and phases minor single nucleotide substitution variants in complex populations. It identifies codon-wise variants in coding regions, performs a reference-guided, *de novo* variant discovery, and annotates known drug-resistance mutations. Insertion and deletion variants are currently ignored; support will be added in a future version. There is no technical limitation with respect to the target organism or gene.

The underlying model is a statistical test, the Bonferroni-corrected Fisher's Exact test. It compares the number of observed mutated codons to the number of expected mutations at a given position.

`juliet` uses JSON target configuration files to define different genes in longer reference sequences, such as overlapping open reading frames in HIV. These predefined configurations ease batch applications and allow immediate reproducibility. A target configuration may contain multiple coding regions within one reference sequence and optional drug resistance mutation positions.

- **Note**: The preinstalled target configurations are meant for a quick start. It is the user's responsibility to ensure that the target configurations used are correct and up-to-date.
- **Note**: If the target configuration `none` was specified, the provided reference is expected to be in-frame.

Performance

At a coverage of 6000 CCS reads with a predicted accuracy (RQ) of ≥`0.99`, the false positive and false negative rates are below 1% and 0.001% (10^{-5}), respectively.

Usage

```
$ juliet --config "HIV" data.align.bam patientZero.html
```

Required	Description
input_file.bam	Input aligned BAM file containing CCS records, which must be PacBio-compliant, that is, `cigar M` is forbidden.

Required	Description
`output_file.html`	Output report HTML file.

Configuration	Description
`--config,-c`	Path to the target configuration JSON file, predefined target configuration tag, or the JSON string.
`--mode-phasing,-p`	Phase variants and cluster haplotypes.

Restrictions	Description
`--region,-r`	Specify the genomic region of interest; reads will be clipped to that region. Empty means **all** reads.
`--drm-only,-k`	Only report DRM positions specified in the target configuration. Can be used to filter for drug-resistance mutations - only known variants from the target configuration will be called.
`--min-perc,-m`	The minimum variant percentage to report. Example: `--min-perc 1` will only show variant calls with an observed abundance of more than 1%. (Default = `0`)
`--max-perc,-n`	The maximum variant percentage to report. Example: `--max-perc 95` will only show variant calls with an observed abundance of less than 95%. (Default = `100`)

Chemistry Override (Specify both)	Description
`--sub,-s`	Substitution rate. Specify to override the learned rate. (Default = `0`)
`--del,-d`	Deletion rate. Specify to override the learned rate. (Default = `0`)

Options	Description
`--help, -h`	Displays help information and exits.
`--verbose, -v`	Sets the verbosity level.
`--version`	Displays program version number and exits.
`--debug`	Returns all amino acids, irrespective of their relevance.
`--emit-tool-contract`	Emits the tool contract.
`--resolved-tool-contract`	Use arguments from the resolved tool contract.
`--mode-phasing,-p`	Phase variants and cluster haplotypes.

Input Files

- BAM-format files containing CCS records. These must be PacBio-compliant, that is, `cigar M` is forbidden.
- Input CCS reads should have a minimal predicted accuracy of `0.99`.
- Reads should be created with CCS2 using the `--richQVs` option. Without the `--richQVs` information, the number of false positive calls might be higher, as `juliet` is missing information to filter actual heteroduplexes in the sample provided.

- `Juliet` currently does **not** demultiplex barcoded data; you must provide one BAM file per barcode.

Output Files

A JSON and/or HTML file:

```
$ juliet data.align.bam patientZero.html
$ juliet data.align.bam patientZero.json
$ juliet data.align.bam patientZero.html patientZero.json
```

The HTML file includes the same content as the JSON file, but in more human-readable format. The HTML file contains four sections:

1. Input Data

Summarizes the data provided, the exact call for `juliet`, and `juliet` version for traceability purposes.

2. Target Config

Summarizes details of the provided target configuration for traceability. This includes the configuration version, reference name and length, and annotated genes. Each gene name (in bold) is followed by the reference start, end positions, and possibly known drug resistance mutations.

▼ **Target config**

Config Version: `Predefined v1.1, PacBio internal`
Reference Name: `HIV HXB2`
Reference Length: `9719`
Genes:

- **5'LTR** (1-634)
- **p17** (790-1186)
- **p24** (1186-1879)
- **p2** (1879-1921)
- **p7** (1921-2086)
- **p1** (2086-2134)
- **p6** (2134-2292)
- **Protease** (2253-2550)
  - `ATV/r: V32I L33F M46I M46L I47V G48V G48M I50L I54V I54T I54A I54L I54M V82A V82T V82F V82S I84V N88S L90M`
  - `DRV/r: V32I L33F I47V I47A I50V I54L I54M L76V V8F I84V`
  - `FPV/r: V32I L33F M46I M46L I47V I47A I50V I54V I54T I54A I54L I54M L76V V82A V82T V82F V82S I84V L90M`
  - `IDV/r: V32I M46I M46L I47V I54V I54T I54A I54L I54M L76V V82A V82T V82F V82S I84V N88S L90M`
  - `NFV: D30N L33F M46I M46L I47V G48V G48M I54V I54T I54A I54L I54M V82A V82T V82F V82S I84V N88D N88S L90M`
  - `SQV/r: G48V G48M I54V I54T I54A I54L I54M V82A V82T I84V N88S L90M`
  - `TPV/r: V32I L33F M46I M46L I47V I47A I54V I54A I54M V82T V82L I84V`

3. Variant Discovery

For each gene/open reading frame, there is one overview table.

Each row represents a variant position. Each variant position consists of the reference codon, reference amino acid, relative amino acid position in the gene, mutated codon, percentage, mutated amino acid, coverage, and possible affected drugs.

Clicking the row displays counts of the multiple-sequence alignment counts of the -3 to +3 context positions.

▼ Variant Discovery

Reverse Transcriptase							
HIV HXB2			Sample Variants				
Codon	AA	Pos	AA	Codon	%	Coverage	Affected Drugs*
ATG	M	41	L	TTG	1	2793	ABC + DDI + TDF + D4T + ZDV
AAA	K	65	R	AGA	1.1	2529	3TC + FTC + ABC + DDI + TDF + D4T

Pos	A	C	G	T	-	N
-3	**2947**	0	0	0	0	51
-2	**2923**	0	2	0	0	73
-1	4	0	**2952**	0	0	42
0	**2606**	0	0	0	339	53
1	**2905**	0	29	0	0	64
2	**2938**	0	0	0	0	60
3	**2938**	0	0	0	0	60
4	**2942**	0	0	0	0	56
5	**2751**	0	0	0	0	247

Codon	AA	Pos	AA	Codon	%	Coverage	Affected Drugs*
TAT	Y	181	C	TGT	0.91	2946	NVP + EFV + ETR + RPV
GGA	G	190	A	GCA	1	2947	NVP + EFV + ETR + RPV
ACC	T	215	Y	TAC	0.93	2877	ABC + DDI + TDF + D4T + ZDV

*HIVdb version 8.3 (last updated 2017-03-02)

▶ Legend

4. Drug Summaries

Summarizes the variants grouped by annotated drug mutations:

▼ Drug Summaries

		Reference		Sample	
Drug	Gene	AA	Pos	AA	%
3TC	Reverse Transcriptase	K	65	R	1
ABC	Reverse Transcriptase	M	41	L	0.99
		K	65	R	1
		T	215	Y	0.88

Predefined Target Configuration

`juliet` ships with one predefined target configuration, for HIV. The following is the command syntax for running that predefined target configuration:

```
$ juliet --config "HIV" data.align.bam patientZero.html
```

p6							
HIV HXB2			Sample Variants				
Codon	AA	Pos	AA	Codon	%	Coverage	Affected Drugs*
AAC	N	47	S	AGT	0.95	2924	

Protease							
HIV HXB2			Sample Variants				
Codon	AA	Pos	AA	Codon	%	Coverage	Affected Drugs*
CGA	R	8	X	TGA	0.98	2931	

Reverse Transcriptase							
HIV HXB2			Sample Variants				
Codon	AA	Pos	AA	Codon	%	Coverage	Affected Drugs*
ATG	M	41	L	TTG	0.99	2903	ABC + DDI + TDF + D4T + ZDV
AAA	K	65	R	AGA	1	2577	3TC + FTC + ABC + DDI + TDF + D4T
TTA	L	100	F	TTT	0.85	2819	
TAT	Y	181	C	TGT	0.95	2939	NVP + EFV + ETR + RPV
GGA	G	190	A	GCA	1	2941	NVP + EFV + ETR + RPV
ACC	T	215	Y	TAC	0.88	2940	ABC + DDI + TDF + D4T + ZDV

- **Note**: For the predefined configuration `HIV`, please use the HIV HXB2 complete genome for alignment.

Customized Target Configuration

To define your own target configuration, create a JSON file. The root child `genes` contains a list of coding regions, with `begin` and `end`, the name of the gene, and a list of drug resistant mutations. Each DRM consists of its name and the positions it targets. The `drms` field is optional. If provided, the `referenceSequence` is used to call mutations, otherwise it will be tested against the major codon. All indices are with respect to the provided alignment space, 1-based, begin-inclusive and end-exclusive `[)`.

Target Configuration Example 1- A customized `json` target configuration file named `my_customized_hiv.json`:

```
{
    "genes": [
        {
            "begin": 2550,
            "drms": [
                {
                    "name": "fancy drug",
                    "positions": [ "M41L" ]
```

```
                }
            ],
            "end": 2700,
            "name": "Reverse Transcriptase"
        }
    ],
    "referenceName": "my seq",
    "referenceSequence": "TGGAAGGGCT...",
    "version": "Free text to version your config files"
    "databaseVersion": "DrugDB version x.y.z (last updated YYYY-MM-DD)"
}
```

Run with a customized target configuration using the `--config` option:

```
$ juliet --config my_customized_hiv.json data.align.bam
patientZero.html
```

Valid Formats for DRMs/positions

`103`	**Only** the reference position.
`M130`	Reference amino acid and reference position.
`M103L`	Reference aa, reference position, mutated aa.
`M103LKA`	Reference aa, reference position, list of possible mutated aas.
`103L`	Reference position and mutated aa.
`103LG`	Reference position and list mutated aas.

Missing amino acids are processed as wildcard (`*`).

Example:

```
{ "name": "ATV/r", "positions": [ "V32I", "L33", "46IL",
"I54VTALM", "V82ATFS", "84" ] }
```

Target Configuration Example 2 - BCR-ABL:

For BCR-ABL, using the ABL1 gene with the following reference NM_005157.5 (**https://www.ncbi.nlm.nih.gov/nuccore/NM_005157.5**) a typical target configuration could look like this:

```
{
    "genes": [
        {
            "name": "ABL1",
            "begin": 193,
            "end": 3585,
            "drms": [
                {
                    "name": "imatinib",
                    "positions": [
                      "T315AI","Y253H","E255KV","V299L","F317AICLV","F359CIV" ]
                },
                {
                    "name": "dasatinib",
                    "positions": [ "T315AI","V299L","F317AICLV" ]
                },
                {
                    "name": "nilotinib",
                    "positions": [ "T315AI","Y253H","E255KV","F359CIV" ]
                },
                {
                    "name": "bosutinib",
```

```
                "positions": [ "T315AI" ]
            }
        ]
    }
  ],
"referenceName": "NM_005157.5",
  "referenceSequence": "TTAACAGGCGCGTCCC..."
```

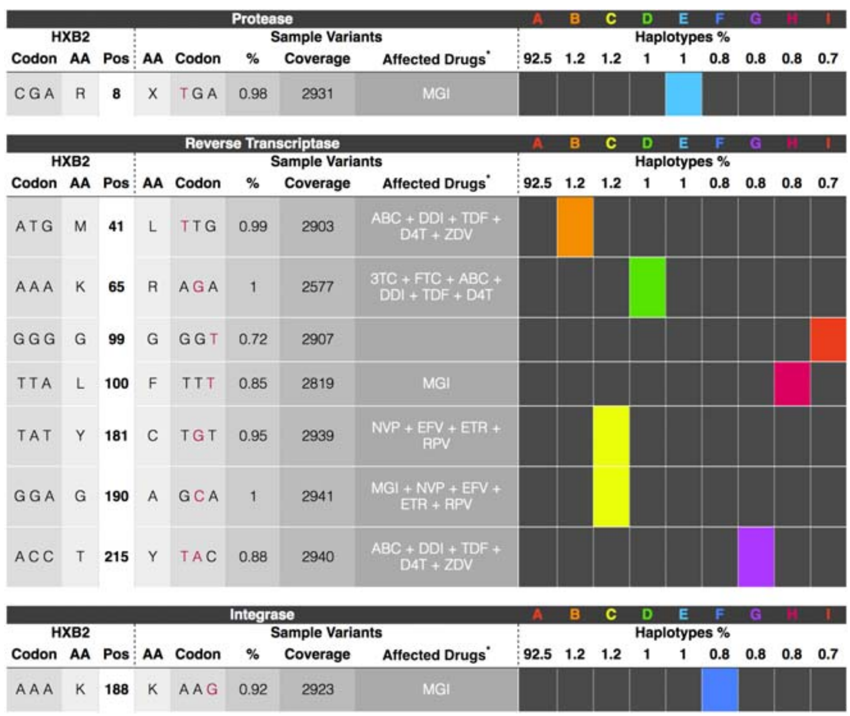

No Target Configuration

If **no** target configuration is specified, either make sure that the sequence is in-frame, or specify the region of interest to mark the correct reading frame, so that amino acids are correctly translated. The output will be labeled with `unknown` as the gene name:

```
$ juliet data.align.bam patientZero.html
```

Phasing

The default mode is to call amino-acid/codon variants independently. Using the `--mode-phasing` option, variant calls from distinct haplotypes are clustered and visualized in the HTML output.

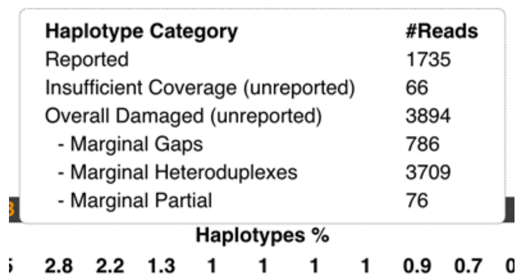


- The row-wise variant calls are "transposed" onto per-column haplotypes. Each haplotype has an ID: `[A-Z]{1}[a-z]?`.
- For each variant, colored boxes in this row mark haplotypes that contain this variant.
- Colored boxes per haplotype/column indicate variants that co-occur. Wild type (no variant) is represented by plain dark gray. A color palette helps to distinguish between columns.

- The JSON variant positions has an additional `haplotype_hit` boolean array with the length equal to the number of haplotypes. Each entry indicates if that variant is present in the haplotype. A haplotype block under the root of the JSON file contains counts and read names. The order of those haplotypes matches the order of all `haplotype_hit` arrays.

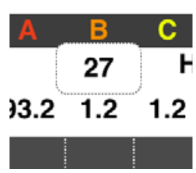
There are two types of tooltips in the haplotype section of the table.

The first tooltip is for the **Haplotypes %** and shows the number of reads that count towards (a) actually reported haplotypes, (b) haplotypes that have less than 10 reads and are not being reported, and (c) haplotypes that are not suitable for phasing. Those first three categories are mutually exclusive and their sum is the total number of reads going into `juliet`. For (c), the three different marginals provide insights into the sample quality; as they are marginals, they are not exclusive and can overlap. The following image shows a sample with bad PCR conditions:



Haplotype Category	#Reads
Reported	1735
Insufficient Coverage (unreported)	66
Overall Damaged (unreported)	3894
- Marginal Gaps	786
- Marginal Heteroduplexes	3709
- Marginal Partial	76

The second type of tooltip is for each haplotype percentage and shows the number of reads contributing to this haplotype:



laa Long Amplicon Analysis (LAA) finds phased consensus sequences from a pooled set of (possibly polyploid) amplicons sequenced with Pacific Biosciences' SMRT technology. Sometimes referred to as **LAA2**, the executable `laa` is a complete rewrite of the `AmpliconAnalysis` module from the `ConsensusTools` package included with earlier versions of SMRT Analysis, which performed a similar function in the Quiver framework. This is a computational and memory-intensive software tool that builds upon the Arrow framework for generating high-quality consensus sequences. It is generally preferable to run LAA using the SMRT Link interface for efficient distribution across a compute cluster. However, it is occasionally useful to run LAA from the command-line to identify optimal parameter settings or to diagnose a problem.

Run Modes

`AmpliconAnalysis` is a general solution for the analysis of PCR products generated with SMRT sequencing, and it can be run in multiple configurations depending on the design of the experiment.

1. **Pooled Polyploid Amplicons**: The default mode assumes that the data contains a single complex mixture of amplicons, which may come from different genes and may have multiple alleles.
2. **Barcoded Polyploid Amplicons:** If passed a file of barcoding results, `AmpliconAnalysis` will instead separate the data by barcode and run the above process on each subset.
3. **Barcoded Simple Amplicons:** Another common use case is to generate consensus sequences for a large number of simple amplicons, such as for synthetic construct validation or high-throughput screening.

Input Files

Long Amplicon Analysis **only** accepts PacBio-compatible BAM files or Data Set XML files as input.

- If your data was generated on a PacBio *RS* or PacBio RS II instrument, see "bax2bam" on page 5 for details on how to convert older data to the new file formats.

In addition, the underlying files themselves now contain barcode information. This document assumes that you already have a barcoded PacBio BAM file containing the data to be analyzed.

Output Files

LAA produces two sets of FASTQ files containing a sequence for each phased template sequence in each coarse cluster, and for each barcode.

- `amplicon_analysis.fastq`: Contains all of the high-quality non-artifactual sequences found.
- `amplicon_analysis_chimeras_noise.fastq`: Contains sequences thought to be some form of PCR or sequencing artifact.

  Note: A sequence is defined as an artifact if, in the summary CSV file, the value of either the `IsDuplicate`, `NoiseSequence` or `IsChimera` columns are `True`.
- `amplicon_analysis_summary.csv`: Contains summary information about each read. Empty fields and values of `-1` represent inapplicable columns, while fields with `1` represent `True` and `0` represents `False`. Contains the following fields:
  - `BarcodeName`: Name of the barcode the reads came from. This is set to `0` for non-barcode runs.
  - `FastaName`: Sequence ID or header string.
  - `CoarseCluster`: Number of the coarse cluster the sequence came from.
  - `Phase`: Number of the phase of the sequence in the coarse cluster.

- `TotalCoverage`: Total number of subreads mapped to this sequence. This may be capped using the `numPhasingReads` option.
- `SequenceLength`: Length of this consensus sequence.
- `ConsensusConverged`: `1` if a final consensus was reached within the alloted iterations, `0` if otherwise. `0` may indicate problems with the underlying sample or data.
- `PredictedAccuracy`: Predicted accuracy of the consensus sequence, calculated by multiplying together the QVs generated by Arrow.
- `NoiseSequence`: `1` if the sequence has a low-quality consensus, corresponding to a predicted accuracy less than 95% indicating a probable PCR artifact; otherwise `0`.
- `IsDuplicate`: `1` if the sequence is a duplicate of another with more coverage, otherwise `0`.
- `DuplicateOf`: If `IsDuplicate` is `1`, contains the name of the other sequence, otherwise empty.
- `IsChimera`: `1` if the sequence is tagged as a chimeric by the UCHIME-like chimera labeler, `0` if otherwise.
- `ChimeraScore`: UCHIME-like score for sequences tested as possible chimeras.
- `ParentSequenceA`: If chimeric, the name of the consensus thought to be the source of the left half.
- `ParentSequenceB`: If chimeric, the name of the consensus thought to be the source of the right half.
- `CrossoverPosition`: Position in the chimeric sequence where the junction between the parent sequences is thought to have occurred.

- `amplicon_analysis_subreads.X.csv`: Contains mapping probabilities for each subread used to call the consensus sequences generated. A **separate** file is written for **each** barcode pair, where `X` is replaced with the name of that pair. Contains the following fields:
  - `SubreadId`: The name of a particular subread used in the current run.
  - `<A Consensus Sequence Name>`: The mapping probability for the subread listed in `SubreadId` to the particular consensus sequence named.

Usage

```
laa [options] INPUT
```

Options	Description
`-h, --help`	Displays help information and exits.
`--verbose, -v`	Sets the verbosity level.
`--version`	Displays program version number and exits.
`--log level`	Sets the logging level. (Default = `INFO`)
`--rngSeed`	RNG seed, modulates reservoir filtering of reads. (Default = `42`)
`--generateBamIndex`	Generate PacBio indicies (`*.pbi`) for BAM files that don't have them.
`--ignoreBamIndex`	Ignore PacBio indicies (`*.pbi`) for BAM files if they exist.
`-M,--modelPath`	Path to a model file or directory containing model files.
`-m,--modelSpec`	Name of chemistry or model to use, overriding the default selection.

Options	Description
`--numThreads,-n`	Number of threads to use; `0` means autodetection. (Default = `0`)
`--takeN`	Report only the top `N` consensus sequences for each barcode. To **disable**, use a number less than 1. (Default = `0`)
`-t,--trimEnds`	Trim `N` bases from each end of each consensus. (Default = `0`)
`--minPredictedAccuracy`	Minimum predicted consensus accuracy below which a consensus is treated as noise. (Default = `0.949999988079071`)
`--chimeraScoreThreshold`	Minimum score to consider a sequence chimeric. (Default = `1`)
`--ChimeraFilter`	Activate the chimera filter and separate chimeric consensus outputs.
`--noChimeraFilter`	Deactivate the chimera filter and output all consensus.
`--logFile`	Output file to write logging information to.
`--resultFile`	Output file name for high-quality results. (Default = `amplicon_analysis.fastq`)
`--junkFile`	Output file name for low-quality or chimeric results. (Default = `amplicon_analysis_chimeras_noise.fastq`)
`--reportFile`	Output file name for the summary report. (Default = `amplicon_analysis_summary.csv`)
`--inputReportFile`	Output file name for the output estimates of input PCR quality, based on subread mappings. (Default = `amplicon_analysis_input.csv`)
`--subreadsReportPrefix`	Prefix for the output subreads report. (Default = `amplicon_analysis_subreads`)
`-b,--barcodes`	FASTA file name of the barcode sequences used, which **overwrites** any barcode names in the Data Set. **Note**: This is used **only** to find barcode names.
`--minBarcodeScore`	Minimum average barcode score required for subreads. (Default = `0`)
`--fullLength`	Filter input reads by presence of both flanking barcodes.
`--doBc`	A comma-separated list of barcode pairs to analyse. This can be by name ("`lbc1--lbc1`") or by Index ("`0--0`").
`--ignoreBc`	Disable barcode filtering so that all data be treated as one sample.
`-l,--minLength`	Minimum length of input reads to use. (Default = `3000`)
`-L,--maxLength`	Maximum length of input reads to use. To **disable**, set to `0`. (Default = `0`)
`-s,--minReadScore`	Minimum read score of input reads to use. (Default = `0.75`)
`--minSnr`	Minimum SNR of input reads to use. (Default = `3.75`)
`--whitelist`	A file of ReadIds, in either Text or FASTA format, to allow from the input file. (Default = `NONE`)
`-r,--maxReads`	Maximum number of input reads, per barcode, to use in analysis. (Default = `2000`)
`-c,--maxClusteringReads`	Maximum number of input reads to use in the initial clustering step. (Default = `500`)
`--fullLengthPreference`	Prefer full-length subreads when clustering.
`--fullLengthClustering`	Only use full-length subreads when clustering.
`--clusterInflation`	Markov clustering inflation parameter. (Default = `2`)
`--clusterLoopWeight`	Markov clustering loop weight parameter. (Default = `0.00100000004749745`)

Options	Description
`--skipRate`	Skip some high-scoring alignments to disperse the cluster more. (Default = `0.0`)
`--minClusterSize`	Minimum number of reads supporting a cluster before it is reported. (Default = `20`)
`--doCluster`	Only analyze one specified cluster. (Default = `-1`)
`--Clustering`	Enable coarse clustering.
`--noClustering`	Disable coarse clustering.
`-i,--ignoreEnds`	When splitting, ignore `N` bases at the end. This prevents excessive splitting caused by degenerate primers. (Default = `0`)
`--maxPhasingReads`	Maximum number of reads to use for phasing/consensus. (Default = `500`)
`--minQScore`	Minimum score to require of mutations used for phasing. (Default = `20`)
`--minPrevalence`	Minimum prevalence to require of mutations used for phasing. (Default = `0.0900000035762787`)
`--minSplitReads`	Minimum number reads favoring the minor phase required to split a haplotype. (Default = `20`)
`--minSplitFraction`	Minimum fraction of reads favoring the minor phase required to split a haplotype. (Default = `0.100000001490116`)
`--minSplitScore`	The global likelyhood improvement required to split a haplotype. (Default = `500`)
`--minZScore`	Minimum Z Score to allow before adding a read to a haplotype. (Default = `-10`)
`--Phasing`	Enable the fine phasing step.
`--noPhasing`	Disable the fine phasing step.
`--emit-tool-contract`	Emits the tool contract.
`--resolved-tool-contract`	Use arguments from the resolved tool contract.

Algorithm Description

LAA proceeds in six main phases: Data filtering, coarse clustering, waterfall clustering, fine phasing, consensus polishing, and post-processing.

- **Data filtering** is used to separate out sequences by their barcode calls, if present, so that only reads long enough to meaningfully contribute to phasing are used.
- The **Coarse and Waterfall Clustering** steps are used to find and separate reads coming from different amplicons.
- The reads from each cluster are then put through the **phasing** step, which recursively separates full-length haplotypes using a variant of the Arrow model. Those haplotypes are then **polished** within the Arrow framework to achieve a high-quality consensus sequence.
- Finally, a **post-processing** step attempts to identify and remove spurious consensus sequences and sequences representing PCR artifacts.

Data Filtering

In this first step, we separate sequences by barcode and then apply a series of user-selected quality filters to speed up down-stream processing and improve result quality. Filters are used primarily to remove short subreads (which may not be long enough to phase variants of interest) and subreads with low barcode scores (representing reads for whom the barcode call is uncertain and may be incorrect). A “Whitelist” option is also available so that users can specify the exact list of subreads or ZMWs to use.

Coarse Clustering and Waterfall Clustering

The coarse clustering step groups the number of subreads (set by the `maxClusteringReads` option) that originate from different amplicons into different clusters. It works by detecting subread-to-subread similarities, building a graph of the results, and then clustering nodes (subreads) using the Markov Clustering algorithm (**http://micans.org/mcl/**). The Markov clustering step is needed to remove spurious similarities caused by chimeric reads that can arise from PCR errors or doubly-loaded ZMWs, or just by chance due to sequencing error.

Next, if the number of subreads specified with the `maxReads` option is greater than the number used in coarse clustering, any remaining subreads are aligned to a rough consensus of each cluster and added to the cluster with the greatest similarity. This “waterfall” step allows for a larger number of reads to be used much more quickly than if all subreads had to be clustered using the normal coarse clustering process.

At the end of clustering, subreads in each cluster are then sorted for downstream analysis using the PageRank algorithm (Page, Lawrence, et al. “The PageRank citation ranking: Bringing order to the web.” (1999)). This ensures that the most representative reads of the cluster are used first in the generation of consensus sequences.

Phasing/Consensus

The reads assigned to each cluster are loaded into the Arrow framework, and an initial consensus of all reads is found. SNP differences between subreads and the initial consensus are scored with the Arrow model, and combinations of high-scoring SNPs are tested for their ability to segregate the reads into multiple haplotypes. If sufficient evidence of a second haplotype is found, the template sequence is ‘split’ into two copies, one with the SNPs applied to the template and one without. This process is repeated recursively so long as new haplotypes with sufficient scores can be found with at least some minimum level of coverage.

Post-Processing Filters

LAA implements a post-processing step to flag likely PCR artifacts in the set of phased output sequences. First, consensus sequences that are identical duplicates of other consensus sequences in the results are removed. Next, those with unusually low predicted accuracy are flagged as being probable sequencing artifacts and removed. We implemented a filter for consensus sequences from PCR crossover events, which on average make up ~5 to 20% of products generated by PCR amplifications >3 kb in length.

For artifacts of PCR crossover events, or “chimeras”, we implemented a variant of the UCHIME algorithm (Edgar, Robert C., et al. “UCHIME improves sensitivity and speed of chimera detection.” Bioinformatics 27.16(2011): 2194-2200). The consensus sequences are sorted in order of decreasing read coverage, and the first two sequences are accepted as non-chimeric since they have no possible parent sequences with greater coverage. The remaining sequences are evaluated in descending order, with **each** test sequence aligned to all non-chimeric sequences so far processed. Crossovers between pairs of non-chimeric sequences are checked to see if they would yield a sequence very similar to the test sequence. If one is found with a sufficient score, the test sequence is marked as chimeric. If not, the test sequence is added to the list of non-chimeric sequences.

motifMaker

The `motifMaker` tool identifies motifs associated with DNA modifications in prokaryotic genomes. Modified DNA in prokaryotes commonly arises from restriction-modification systems that methylate a specific base in a specific sequence motif. The canonical example is the m6A methylation of adenine in GATC contexts in *E. coli*. Prokaryotes may have a very large number of active restriction-modification systems present, leading to a complicated mixture of sequence motifs.

PacBio® SMRT sequencing is sensitive to the presence of methylated DNA at single base resolution, via shifts in the polymerase kinetics observed in the real-time sequencing traces. For more background on modification detection, see **http://nar.oxfordjournals.org/content/early/2011/12/07/nar.gkr1146.full.**

Algorithm

Existing motif-finding algorithms such as MEME-chip and YMF are sub-optimal for this case for the following reasons:

- They search for a **single** motif, rather than attempting to identify a complicated mixture of motifs.
- They generally don't accept the notion of aligned motifs - the input to the tools is a window into the reference sequence which can contain the motif at any offset, rather than a single center position that is available with kinetic modification detection.
- Implementations generally either use a Markov model of the reference (MEME-chip), or do exact counting on the reference, but place restrictions on the size and complexity of the motifs that can be discovered.

Following is a rough overview of the algorithm used by `motifMaker`: Define a motif as a set of tuples: (Position relative to methylation, required base.) Positions not listed in the motif are implicitly degenerate. Given a list of modification detections and a genome sequence, we define the following objective function on motifs:

```
Motif score(motif) = (# of detections matching motif) / (# of
genome sites matching motif) * (Sum of log-pvalue of detections
matching motif) = (fraction methylated) * (sum of log-pvalues of
matches)
```

We search (close to exhaustively) through the space of all possible motifs, progressively testing longer motifs using a branch-and-bound search. The 'fraction methylated' term must be less than 1, so the maximum achievable score of a child node is the sum of scores of modification hits in the current node, allowing pruning of all search paths whose maximum achievable score is less than the best score discovered so far.

Usage

For command-line motif-finding, run the `find` command, and pass the reference FASTA and the `modifications.gff` (.gz) file output by the PacBio modification detection workflow.

The `reprocess` subcommand annotates the GFF file with motif information for better genome browsing.

```
MotifMaker [options] [command] [command options]
```

`find` Command: Run motif-finding.

```
find [options]
```

Options	Description
`-h, --help`	Displays help information and exits.
`* -f, --fasta`	Reference FASTA file.
`* -g, --gff`	`Modifications.gff` or `.gff.gz` file.
`-m, --minScore`	Minimum Qmod score to use in motif finding. (Default = `40.0`)
`* -o, --output`	Output `motifs.csv` file.
`-x, --xml`	Output motifs XML file.

`reprocess` Command: Update a `modifications.gff` file with motif information based on new Modification QV thresholds.

```
reprocess [options]
```

Options	Description
`-c, --csv`	Raw `modifications.csv` file.

Options	Description
`* -f, --fasta`	Reference FASTA file.
`* -g, --gff`	`Modifications.gff` or `.gff.gz` file.
`-m, --minFraction`	Only use motifs above this methylated fraction. (Default = `0.75`)
`-m, --motifs`	`Motifs.csv` file.
`* -o, --output`	Reprocessed `modifications.gff` file.

Output Files

Using the `find` command:

- **Output CSV file**: This file has the same format as the standard "Fields included in motif_summary.csv" described in the Methylome Analysis White Paper (**https://github.com/PacificBiosciences/Bioinformatics-Training/wiki/Methylome-Analysis-Technical-Note**).

Using the `reprocess` command:

- **Output GFF file**: The format of the output file is the same as the input file, and is described in the Methylome Analysis White Paper (**https://github.com/PacificBiosciences/Bioinformatics-Training/wiki/Methylome-Analysis-White-Paper**) under "Fields included in the modifications.gff file".

pbalign

The `pbalign` tool aligns PacBio reads to reference sequences; filters aligned reads according to user-specific filtering criteria; and converts the output to PacBio BAM, SAM, or PacBio DataSet format.

Input Files

The `pbalign` tool distinguishes input and output file formats by file extensions. The tool supports the following input formats:

- BAM: `.bam`
- DataSet: `.subreadset.xml` or `.consensusreadset.xml`
- FASTA: `.fa` or `.fasta`
- File-Of-File-Names: `.fofn`

The input reference sequences can be in a FASTA file or a reference dataset created by `fasta-to-reference`, a PacBio tool for converting references in a FASTA file to a PacBio reference dataset. See “fasta-to-reference” on page 30 for details.

Output Files

The tool supports the following output formats:

- BAM: `.bam`

- DataSet: `.xml`
- SAM: `.sam`

Usage

```
pbalign [-h] [--verbose] [--version] [--profile] [--debug]
        [--regionTable REGIONTABLE] [--configFile CONFIGFILE]
        [--algorithm {blasr,bowtie}] [--maxHits MAXHITS]
        [--minAnchorSize MINANCHORSIZE]
        [--maxMatch MAXMATCH]
        [--useccs {useccs,useccsall,useccsdenovo}]
        [--noSplitSubreads] [--nproc NPROC]
        [--algorithmOptions ALGORITHMOPTIONS]
        [--maxDivergence MAXDIVERGENCE] [--minAccuracy MINACCURACY]
        [--minLength MINLENGTH]
        [--scoreFunction {alignerscore,editdist,blasrscore}]
        [--scoreCutoff SCORECUTOFF]
        [--hitPolicy {randombest,allbest,random,all}] [--forQuiver]
        [--seed SEED] [--tmpDir TMPDIR]
        inputFileName referencePath outputFileName
```

Required	Description
`inputFileName`	The input file of PacBio reads. Can be a BAM, Data Set, FASTA file, or a `fofn` (File-Of-File-Names).
`referencePath`	Either a reference FASTA file or a PacBio reference dataset file.
`outputFileName`	The output `.bam`, `.xml` or `.sam` file.

Options	Description
`-h, --help`	Displays help information and exits.
`--verbose, -v`	Sets the verbosity level.
`--version`	Displays program version number and exits.
`--profile`	Print runtime profile at exit.
`--debug`	Run within a debugger session.
`--configFile`	Specify a set of user-defined argument values.
`--algorithm`	Select an algorithm from `blasr` or `bowtie`. (Default = `blasr`)
`--maxHits`	The maximum number of matches of each read to the reference sequence that will be evaluated. (Default = `10`)
`--minAnchorSize`	The length of the read that must match against the reference sequence. (Default = `12`)
`--maxMatch`	Stop extending an anchor between the read and the reference sequence when its length reaches this value. Bypasses the blasr `maxMatch` option. (Default = `30`)
`--noSplitSubreads`	Do **not** split reads into subreads even if subread regions are available. (Default = `False`)
`--nproc NPROC`	Number of threads. (Default = `8`)
`--algorithmOptions`	Pass alignment options through.
`--maxDivergence`	The maximum allowed percentage divergence of a read from the reference sequence. (Default = `30`)

Options	Description
`--minAccuracy`	The minimum percentage accuracy of alignments that will be evaluated. (Default = `70`)
`--minLength`	The minimum aligned read length of alignments that will be evaluated. (Default = `50`)
`--scoreFunction`	Specify a score function for evaluating alignments. • `alignerscore`: Aligner's score in the SAM tag `as`. • `editdist`: Edit distance between read and reference. • `blasrscore`: The `blasr` default score function. (Default = `alignerscore`)
`--scoreCutoff`	The worst score to output an alignment.
`--hitPolicy`	Specify a policy for how to treat multiple hits. • `random`: Selects a random hit. • `all`: Selects **all** hits. • `allbest`: Selects all the best score hits. • `randombest`: Selects a random hit from all best alignment score hits. (Default = `randombest`)
`--seed`	Initialize the random number generator with a non-zero integer. `0` means that current system time is used. (Default = `1`)
`--tmpDir`	Specify a directory for saving temporary files. (Default = `/scratch`)

Examples

Basic usage:

```
$ pbalign tests/data/example/read.bam   \
          tests/data/example/ref.fasta  \
          tests/data/example/example.bam
```

Basic usage with optional arguments:

```
$ pbalign --maxHits 10 --hitPolicy all  \
            tests/data/example_read.fasta \
            tests/data/example_ref.fasta  \
            example.sam
```

Advanced usage - To import predefined options from a configuration file:

```
$ pbalign --configFile=tests/data/1.config \
            tests/data/example/read.fasta    \
            tests/data/example/ref.fasta     \
            example.sam
```

Advanced usage - To pass options through to the Aligner:

```
$ pbalign --algorithmOptions='-nCandidates 10 -sdpTupleSize 12'\
            tests/data/example/read.fasta \
            tests/data/example/ref.fasta   \
            example.sam                   \
```

Advanced usage - To use `pbalign` as a library using the Python API:

```
$ python
>>> from pbalign.pbalignrunner import PBAlignRunner
>>> # Specify arguments in a list.
>>> args = ['--maxHits', '20', 'tests/data/example/read.fasta',\
...         'tests/data/example/ref.fasta', 'example.sam']
>>> # Create a PBAlignRunner object.
>>> a = PBAlignRunner(args)
>>> # Execute.
>>> exitCode = a.start()
>>> # Show all files used.
>>> print a.fileNames
```

pbdagcon

The `pbdagcon` tool implements DAGCon (Directed Acyclic Graph Consensus), which is a sequence consensus algorithm based on using directed acyclic graphs to encode multiple sequence alignments.

`pbdagcon` uses the alignment information from `blasr` to align sequence reads to a "backbone" sequence. Based on the underlying alignment directed acyclic graph (DAG), it will be able to use the new information from the reads to find the discrepancies between the reads and the "backbone" sequences. A dynamic programming process is then applied to the DAG to find the optimum sequence of bases as the consensus. The new consensus can be used as a new backbone sequence to iteratively improve the consensus quality.

While the code is developed for processing Pacific Biosciences raw sequence data, the algorithm can be used for general consensus purposes. Currently, it only takes FASTA input. For shorter read sequences, one might need to adjust the `blasr` alignment parameters to get the alignment string properly.

Note: This code is **not** an official Pacific Biosciences software release.

Examples

To generate consensus from `blasr` alignments:

This is the most basic use case where one can generate a consensus from a set of alignments by directly using the `pbdagcon` executable.

At the most basic level, `pbdagcon` takes information from `blasr` alignments sorted by target and generates FASTA-formatted corrected target sequences. The alignments from `blasr` can be formatted with either -m 4 or -m 5. For -m 4 format, the alignments **must** be run through a format adapter (`m4topre.py`) to generate suitable input to `pbdagcon`.

The following example shows the simplest way to generate a consensus for one target using `blasr` -m 5 alignments as input:

```
blasr queries.fasta target.fasta -bestn 1 -m 5 -out mapped.m5
pbdagcon mapped.m5 > consensus.fasta
```

To generate corrected reads from `daligner` alignments:

Support for generating consensus from `daligner` output exists as a new executable: `dazcon`. Note that `dazcon` is sensitive to the version of `daligner` used and may crash if using inputs generated by versions other than what is referenced in the submodules.

```
dazcon -ox -j 4 -s subreads.db -a subreads.las > corrected.fasta
```

To correct PacBio reads using HGAP:

This example shows how PacBio reads are corrected in PacBio's "Hierarchichal Genome Assembly Process" (HGAP) workflow. HGAP uses `blasr` -m 4 output.

This example makes use of the `filterm4.py` and `m4topre.py` scripts:

```
# First filter the m4 file to help remove chimeras:
filterm4.py mapped.m4 > mapped.m4.filt

# Next run the m4 adapter script, generating 'pre-alignments':
m4topre.py mapped.m4.filt mapped.m4.filt reads.fasta 24 > mapped.pre

# Finally, correct using pbdagcon, typically using multiple consensus threads:
pbdagcon -j 4 -a mapped.pre > corrected.fasta
```

pbindex

The `pbindex` tool creates an index file that enables random-access to PacBio-specific data in BAM files.

Usage

```
pbindex <input>
```

Options	Description
`-h, --help`	Displays help information and exits.
`--version`	Displays program version number and exits.

Input File

- `*.bam` file containing PacBio data.

Output File

- `*.pbi` index file, with the same prefix as the input file name.

pbservice

The `pbservice` tool performs a variety of useful tasks within SMRT Link.

- To get help for `pbservice`, use `pbservice -h`.

- To get help for a specific `pbservice` command, use `pbservice <command> -h.`

All `pbservice` commands include the following optional parameters:

Options	Description
`--host=http://localhost`	The server host. Override the default with the environmental variable `PB_SERVICE_HOST`.
`--port=8070`	The server port. Override the default with the environmental variable `PB_SERVICE_PORT`.
`--log-file LOG_FILE`	Write the log to file. (Default = `None,` writes to `stdout`.)
`--log-level=INFO`	Specify the log level; values are `[DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL.]` (Default = `INFO`)
`--debug=False`	Alias for setting the log level to `DEBUG`. (Default = `False`)
`--quiet=False`	Alias for setting the log level to `CRITICAL` to suppress output. (Default = `False`)

`status` Command**:** Use to get system status.

```
pbservice status [-h] [--host HOST] [--port PORT]
                 [--log-file LOG_FILE]
                 [--log-level INFO}
                 [--debug] [--quiet]
```

`import-dataset` Command**:** Import Local Data Set XML. The file location **must** be accessible from the host where the services are running; often on a shared file system

```
pbservice import-dataset [-h] [--host HOST] [--port PORT]
                         [--log-file LOG_FILE]
                         [--log-level INFO]
                         [--debug] [--quiet]
                         xml_or_dir
```

Required	Description
`xml_or_dir`	Specify a directory or XML file for the Data Set.

`import-fasta` Command**:** Import a FASTA file and convert to a ReferenceSet file. The file location **must** be accessible from the host where the services are running; often on a shared file system.

```
pbservice import-fasta [-h] --name NAME --organism ORGANISM --ploidy
                       PLOIDY [--block] [--host HOST] [--port PORT]
                       [--log-file LOG_FILE]
                       [--log-level INFO]
                       [--debug] [--quiet]
```

```
                       fasta_path
```

Required	Description
fasta_path	Path to the FASTA file to import.

Options	Description
--name	Name of the ReferenceSet to convert the FASTA file to.
--organism	Name of the organism.
--ploidy	Ploidy.
--block=False	Block during importing process.

`run-analysis` **Command:** Run a secondary analysis pipeline using an `analysis.json` file.

```
pbservice run-analysis [-h] [--host HOST] [--port PORT]
                       [--log-file LOG_FILE]
                       [--log-level INFO]
                       [--debug] [--quiet] [--block]
                       json_path
```

Required	Description
json_path	Path to the `analysis.json` file.

Options	Description
--block=False	Block during importing process.

`emit-analysis-template` **Command:** Output an `analysis.json` template to `stdout` that can be run using the `run-analysis` command.

```
pbservice emit-analysis-template [-h] [--log-file LOG_FILE]
                                 [--log-level INFO]
                                 [--debug] [--quiet]
```

`get-job` **Command:** Get a Job Summary by Job Id.

```
pbservice get-job [-h] [--host HOST] [--port PORT]
                  [--log-file LOG_FILE]
                  [--log-level INFO]
                  [--debug] [--quiet]
                  job_id
```

Required	Description
job_id	Job id or UUID.

`get-jobs` **Command:** Get Job Summaries by Job Id.

```
pbservice get-jobs [-h] [-m MAX_ITEMS] [--host HOST] [--port PORT]
                   [--log-file LOG_FILE]
                   [--log-level INFO]
```

```
                   [--debug] [--quiet]
```

Options	Description
`-m=25, --max-items=25`	Maximum number of jobs to get.

`get-dataset` Command**:** Get a Data Set summary by Data Set Id or UUID.

```
pbservice get-dataset [-h] [--host HOST] [--port PORT]
                      [--log-file LOG_FILE]
                      [--log-level INFO]
                      [--debug] [--quiet]
                      id_or_uuid
```

Required	Description
`id_or_uuid`	Data Set Id or UUID.

`get-datasets` Command**:** Get a Data Set list summary by Data Set type.

```
pbservice get-datasets [-h] [--host HOST] [--port PORT]
                       [--log-file LOG_FILE]
                       [--log-level INFO]
                       [--debug] [--quiet] [-m MAX_ITEMS]
                       [-t DATASET_TYPE]
```

Required	Description
`-t=subreads, --dataset-type=subreads`	The type of Data Set to retrieve: `subreads, alignments, references, barcodes`.

`delete-dataset` Command**:** Delete a specified Data Set.
Note: This is a "soft" delete **-** the database record is tagged as inactive so it won't display in any lists, but the files will **not** be removed.

```
pbservice delete-dataset [-h] [--host HOST] [--port PORT]
                         [--log-file LOG_FILE]
                         [--log-level INFO]
                         [--debug] [--quiet]
                         [ID]
```

Required	Description
`ID`	A valid Data Set ID, either UUID or integer ID, for the Data Set to delete.

Examples

To obtain system status, the Data Set summary, and the job summary:

```
pbservice status --host smrtlink-release --port 9091
```

To import a Data Set XML:

```
pbservice import-dataset --host smrtlink-release --port 9091 \
path/to/subreadset.xml
```

To obtain a job summary using the Job Id:

```
pbservice get-job --host smrtlink-release --port 9091 \
--log-level CRITICAL 1
```

To obtain Data Sets by using the Data Set Type `subreads`:

```
pbservice get-datasets --host smrtlink-alpha --port 8081 \
--quiet --max-items 1 -t subreads
```

To obtain Data Sets by using the Data Set Type `alignments`:

```
pbservice get-datasets --host smrtlink-alpha --port 8081 \
--quiet --max-items 1 -t alignments
```

To obtain Data Sets by using the Data Set Type `references`:

```
pbservice get-datasets --host smrtlink-alpha --port 8081 \
--quiet --max-items 1 -t references
```

To obtain Data Sets by using the Data Set Type `barcodes`:

```
pbservice get-datasets --host smrtlink-alpha --port 8081 \
--quiet --max-items 1 -t barcodes
```

To obtain Data Sets by using the Data Set UUID:

```
pbservice get-dataset --host smrtlink-alpha --port 8081 \
--quiet 43156b3a-3974-4ddb-2548-bb0ec95270ee
```

pbsmrtpipe

The `pbsmrtpipe` tool is the secondary analysis workflow engine of Pacific Biosciences' SMRT Analysis software. `pbsmrtpipe` is easily extensible, and supports logging, distributed computing, error handling, analysis parameters, and temporary files.

In a typical installation of the SMRT Analysis Software, SMRT Link's SMRT Analysis module calls `pbsmrtpipe` when an analysis is started. SMRT Link's SMRT Analysis module provides a convenient and user-friendly way to analyze PacBio sequencing data through `pbsmrtpipe`.

For power users, there is more flexibility and customization available by instead running `pbsmrtpipe` analyses from the command line.

The `pbsmrtpipe` command is normally run in one of several modes, which are specified as a positional argument.

For details about a specific pipeline, specify the ID (the last field in each item in the output of `show-templates`) using the `show-template-details` command:

```
$ pbsmrtpipe show-template-details pbsmrtpipe.pipelines.sa3_ds_resequencing
```

Note that if you are starting from PacBio's `bax.h5` basecalling files, you will need to do an initial conversion step.

Pipelines

Following are the available pipelines, their purpose, and their outputs.

Note: All pipeline names are prefixed with `pbsmrtpipe.pipelines`; this is omitted from the table.

Pipeline Name	Description/Common Outputs
`sa3_sat`	• Site Acceptance Test run on all new PacBio installations. • variants GFF, SAT report.
`sa3_ds_resequencing`	• Map subreads to reference genome and determine consensus sequence with Arrow. • AlignmentSet, consensus ContigSet, variants GFF.
`sa3_ds_ccs`	• Generate high-accuracy Circular Consensus Sequences from subreads. • ConsensusReadSet, FASTA and FASTQ files.
`sa3_ds_ccs_mapping`	• ConsensusRead (CCS) + mapping to reference genome, starting from subreads. • ConsensusReadSet, FASTA and FASTQ files, ConsensusAlignment-Set.
`sa3_ds_isoseq_classify`	• Iso-Seq transcript classification, starting from subreads. • ContigSets of classified transcripts.
`sa3_ds_isoseq`	• Full Iso-Seq analysis with clustering and Quiver/Arrow polishing. (This is much slower.) • ContigSets of classified transcripts plus polished isoform ContigSet.
`sa3_ds_isoseq_with_genome`	• Full Iso-Seq analysis with clustering and Quiver/Arrow polishing, followed by collapsing High Quality isoforms against a reference genome to collapse and filter out redundant HQ isoforms and degraded HQ soforms. • ContigSet of classified transcripts, polished HQ/LQ isoform ContigSet, alignments mapping HQ isoforms to a reference genome, collapsed filtered isoform groups in TXT with supportive HQ isoforms in each group, collapsed filtered isoform groups in FASTQ, collapsed filtered isoform groups in GFF which can be viewed in IGV, abundance information of supportive FLNC and NFL reads to collapsed filtered isoform groups.
`sa3_ds_isoseq2`	• Output files of `sa3_ds_isoseq_classify`. • Full Iso-Seq analysis with clustering and Quiver/Arrow polishing.
`sa3_ds_isoseq2_with_genome`	• Full Iso-Seq analysis with clustering and Quiver/Arrow polishing, followed by collapsing High Quality isoforms against a reference genome to collapse and filter out redundant HQ isoforms and degraded HQ Isoforms. • Classified transcripts, polished HQ/LQ isoform in FASTA/FASTQ, alignments mapping HQ isoforms to a reference genome in SAM, collapsed filtered isoform groups in TXT with supportive HQ isoforms in each group, collapsed filtered isoform groups in FASTQ, collapsed filtered isoform groups in GFF which can be viewed in IGV, abundance information of supportive FLNC and NFL reads to collapsed filtered isoform groups.
`ds_modification_motif_analysis`	• Base modification detection and motif-finding, starting from subreads. • Resequencing output plus basemods GFF, motifs CSV.
`sa3_hdfsubread_to_subread`	• Convert HdfSubreadSet to SubreadSet (import bax.h5 basecalling files). • SubreadSet
`sa3_ds_laa`	• Basic Long Amplicon Analysis (LAA) pipeline, from barcoded subreads. • Consensus FASTA

Pipeline Name	Description/Common Outputs
sa3_ds_pbsv	• Structural Variation pipeline. • Reference-aligned reads
polished_falcon_fat	• HGAP 4 assembly pipeline starting from subreads and a configuration file. • ContigSet of assembled contigs
sa3_ds_barcode2	• Demultiplexing of barcoded data; generates one SubreadSet per barcode.

Parallelization

The algorithms used to analyze PacBio data are computationally intensive but also intrinsically highly parallel. `pbsmrtpipe` can scale to at least hundreds of processors on multi-core systems and/or managed clusters. This is handled by two distinct but complementary methods:

- **Multiprocessing** is implemented in the underlying tasks, all of which are generally shared-memory programs. This is effectively always turned on unless the `max_nchunk` parameter is set to `1`. (See the Examples section for a description of how to modify parameter values.) For most compute node configurations, a value between `8` and `16` is appropriate.
- **Parallelization (chunking)** is implemented by `pbsmrtpipe` and works by applying filters to the input Data Sets, which direct tasks to operate on a subset ("chunk") of the data. These chunks are most commonly either a contiguous subset of reads or windows in the reference genome sequence.

Note that the task-level output directories (and the locations of the final result files) may be slightly different depending on whether chunking is used, since an intermediate "gather" step is required to join chunked results.

Usage

```
pbsmrtpipe [-h] [--version]
{pipeline, pipeline-id, task, show-templates, show-template-details, show-tasks, show-task-details, show-workflow-options, run-diagnostic, show-chunk-operators}
                  ...
```

Options	Description
--help	Displays information about command-line usage and options, and then exits. `pipeline-id --help`: Displays information about a specific pipeline.
--version	Displays program version number and exits.

`pipeline` Command: Run a pipeline using a pipeline template or with explicit Bindings and EntryPoints.

```
pbsmrtpipe pipeline [-h] [--debug] -e ENTRY_POINTS [ENTRY_POINTS ...]
                    [-o OUTPUT_DIR] [--preset-xml PRESET_XML]
```

```
[--preset-json PRESET_JSON]
[--preset-rc-xml PRESET_RC_XML]
[--service-uri SERVICE_URI]
[--force-distributed | --local-only]
[--force-chunk-mode | --disable-chunk-mode]
pipeline_template_xml
```

Required	Description
`pipeline_template_xml`	Path to a pipeline template XML file.

Options	Description
`--debug=False`	Alias for setting log level to `DEBUG`.
`-e, --entry`	Entry Points using `entry_idX:/path/to/file.txt` format.
`-o=, --output-dir=`	Path to the job output directory. The directory will be created if it does not exist.
`--preset-xml=[]`	Preset/Option XML file. This option may be repeated if you have multiple preset files.
`--preset-json=[]`	Preset/Option JSON file. This option may be repeated if you have multiple preset files.
`--preset-rc-xml`	Skips loading preset from the environmental variable `PB_SMRTPIPE_XML_PRESET` and explicitly loads the supplied `preset.xml`.
`--service-uri`	Remote Web services to send update and log status to. (This is a JSON file containing the host name and port number.)
`--force-distributed`	Override XML settings to enable distributed mode, if a cluster manager is provided.
`--local-only`	Override XML settings to disable distributed mode.
`--force-chunk-mode`	Override to enable Chunk mode.
`--disable-chunk-mode`	Override to disable Chunk mode.

`pipeline-id` Command: Run a registered pipeline by specifying the pipeline id.

```
pbsmrtpipe pipeline-id [-h] [--debug] -e ENTRY_POINTS
                       [ENTRY_POINTS ...] [-o OUTPUT_DIR]
                       [--preset-xml PRESET_XML]
                       [--preset-json PRESET_JSON]
                       [--preset-rc-xml PRESET_RC_XML]
                       [--service-uri SERVICE_URI]
                       [--force-distributed | --local-only]
                       [--force-chunk-mode | --disable-chunk-mode]
                       pipeline_id
```

Required	Description
`pipeline_id`	Registered pipeline id. Run `show-templates` to display a list of the registered pipelines.

Options	Description
`--debug=False`	Alias for setting log level to `DEBUG`.
`-e, --entry`	Entry Points using `entry_idX:/path/to/file.txt` format.
`-o=, --output-dir=`	Path to the job output directory. The directory will be created if it does not exist.

Options	Description
`--preset-xml=[]`	Preset/Option XML file. This option may be repeated if you have multiple preset files.
`--preset-json=[]`	Preset/Option JSON file. This option may be repeated if you have multiple preset files.
`--preset-rc-xml`	Skips loading preset from the environmental variable `PB_SMRTPIPE_XML_PRESET` and explicitly loads the supplied `preset.xml`.
`--service-uri`	Remote Web services to send update and log status to. (This is a JSON file containing the host name and port number.)
`--force-distributed`	Override XML settings to enable distributed mode, if a cluster manager is provided.
`--local-only`	Override XML settings to disable distributed mode.
`--force-chunk-mode`	Override to enable Chunk mode.
`--disable-chunk-mode`	Override to disable Chunk mode.

`task` Command: Run a task, such as a ToolContract, by id.

```
pbsmrtpipe task [-h] [--debug] -e ENTRY_POINTS [ENTRY_POINTS ...]
                [-o OUTPUT_DIR]
                [--preset-xml PRESET_XML]
                [--preset-json PRESET_JSON]
                [--preset-rc-xml PRESET_RC_XML]
                [--service-uri SERVICE_URI]
                [--force-distributed | --local-only]
                [--force-chunk-mode | --disable-chunk-mode]
                task_id
```

Required	Description
`task_id`	Show details of a registered task by id.

Options	Description
`--debug=False`	Alias for setting log level to `DEBUG`.
`-e, --entry`	Entry Points using `entry_idX:/path/to/file.txt` format.
`-o=, --output-dir=`	Path to the job output directory. The directory will be created if it does not exist.
`--preset-xml=[]`	Preset/Option XML file. This option may be repeated if you have multiple preset files.
`--preset-rc-xml`	Skips loading preset from the environmental variable `PB_SMRTPIPE_XML_PRESET` and explicitly loads the supplied `preset.xml`.
`--service-uri`	Remote Web services to send update and log status to. (This is a JSON file containing the host name and port number.)
`--force-distributed`	Override XML settings to enable distributed mode, if a cluster manager is provided.
`--local-only`	Override XML settings to disable distributed mode.
`--force-chunk-mode`	Override to enable Chunk mode.
`--disable-chunk-mode`	Override to disable Chunk mode.

`show-templates` Command: List all pipeline templates.

A pipeline ‘id’ can be referenced in your `my_pipeline.xml` file using

`<import-template id="pbsmrtpipe.pipelines.my_pipeline_id" />`.

This can replace the explicit listing of EntryPoints and Bindings.

```
pbsmrtpipe show-templates [-h]
                          [--log-level {DEBUG,INFO,WARNING,ERROR,CRITICAL}]
                          [--output-templates-avro OUTPUT_TEMPLATES_AVRO]
                          [--output-templates-json OUTPUT_TEMPLATES_JSON]
```

Options	Description
`--log-level`	Specify the log level; values are [`DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`.]
`--output-templates-avro`	Resolve, validate and output Registered pipeline templates to AVRO files in `output-dir`.
`--output-templates-json`	Resolve, validate and output Registered pipeline templates to JSON files in `output-dir`.

`show-template-details` Command: Displays information about a specific pipeline.

This command lists the entry points required for the pipeline. These are usually PacBio Data Set XML files, although single raw data files (BAM or FASTA format) may be acceptable for some use cases. The most common input will be `eid_subread`, a SubreadSet XML Data Set, which contains one or more BAM files containing the raw unaligned subreads. Also common is `eid_ref_dataset`, for a ReferenceSet or genomic FASTA file.

```
pbsmrtpipe show-template-details [-h] [-o OUTPUT_PRESET_XML]
                                 template_id
```

Required	Description
`template_id`	Show details of a registered Template by id.

Options	Description
`-o, --output-preset-xml`	Write pipeline/task `preset.xml` of options.

`show-tasks` Command: Show completed list of tasks by id.

Use the environmental variable `PB_TOOL_CONTRACT_DIR` to define a custom directory of tool contracts. These tool contracts will override the installed tool contracts, such as `PB_TOOL_CONTRACT_DIR=/path/to/my-tc-dir/`.

```
pbsmrtpipe show-tasks [-h]
```

`show-task-details` Command: Show details of a particular task by id, such as `pbsmrtpipe.tasks.filter_report`.

- Use `show-tasks` to get a complete list of registered tasks.

```
pbsmrtpipe show-task-details [-h] [-o OUTPUT_PRESET_XML] task_id
```

Required	Description
task_id	Show details of a registered task by id.

Options	Description
-o, --output-preset-xml	Write pipeline/task preset.xml of options.

show-workflow-options Command: Display all workflow-level options that can be set in <options /> for preset.xml.

```
pbsmrtpipe show-workflow-options [-h] [-o OUTPUT_PRESET_XML]
```

Options	Description
-o, --output-preset-xml	Write pipeline/task preset.xml ml of options.

run-diagnostic Command: Performs diagnostic tests of preset.xml and the cluster configuration.

```
pbsmrtpipe run-diagnostic [-h] [--debug] [-o OUTPUT_DIR] [--simple]
                          preset_xml
```

Required	Description
preset_xml	Path to Preset XML file.

Options	Description
--debug=False	Alias for setting log level to DEBUG.
-o=, --output-dir=	Path to the job output directory. The directory will be created if it does not exist.
--simple=False	Perform full diagnostics tests; submit a test job to the cluster.

show-chunk-operators Command: Show a list of loaded chunk operators for Scatter/Gather Tasks. Extend resource loading by exporting the environmental variable PB_CHUNK_OPERATOR_DIR.

Example: export PB_CHUNK_OPERATOR_DIR=/path/to/chunk-operators-xml-dir.

```
pbsmrtpipe show-chunk-operators [-h]
```

Example - Basic Resequencing

This pipeline uses pbalign to map reads to a reference genome, and quiver to determine the consensus sequence. The example uses the sa3_ds_resequencing pipeline:

```
$ pbsmrtpipe show-template-details pbsmrtpipe.pipelines.sa3_ds_resequencing
```

This requires two entry points: a SubreadSet and a ReferenceSet. A typical invocation for a hypothetical lambda virus genome might look like this:

```
$ pbsmrtpipe pipeline-id pbsmrtpipe.pipelines.sa3_ds_resequencing \
  -e eid_subread:/data/smrt/2372215/0007/Analysis_Results/\
m150404_101626_42267_c100807920800000001823174110291514_s1_p0.all.subreadset.xml \
  -e eid_ref_dataset:/data/references/lambdaNEB/lambdaNEB.referenceset.xml
```

This will run for a while and output several directories, including tasks, logs, and workflow. The tasks directory is the most useful, as it stores the intermediate results and resolved tool contracts (how the task was executed) for each task. The directory names (`task_ids`) should be somewhat self-explanatory. To direct the output to a subdirectory in the current working directory, use the `-o` option: `-o job_output_1`.

Other pipelines related to resequencing, such as the basemods detection and motif-finding, have nearly identical command-line arguments except for the pipeline ID.

For a general overview of the resequencing results, the GFF file written by `summarizeConsensus` is the most useful:

```
job_output_2/tasks/genomicconsensus.tasks.summarize_consensus-0/
alignment_summary_variants.gff
```

The GFF file contains records for a complete set of sequence regions in the reference genome, including coverage statistics and the number of gaps, substitutions, insertions or deletions. For example:

```
lambda_NEB3011  .       region  1       50      0.00    +       .
cov=116,190,190;cov2=183.000,14.633;gaps=0,0;cQv=20,20,20;del=0;ins=0;sub=0
```

Example - Quiver (Genomic Consensus)

If you already have an AlignmentSet on which you just want to run quiver, the `sa3_ds_genomic_consensus` pipeline will be faster:

```
$ pbsmrtpipe pipeline-id pbsmrtpipe.pipelines.sa3_ds_genomic_consensus \
  -e eid_bam_alignment:/data/project/my_lambda_genome.alignmentset.xml \
  -e eid_ref_dataset:/data/references/lambda.referenceset.xml \
  --preset-xml=preset.xml
```

Example - Circular Consensus Sequences

To obtain high-quality consensus sequences (also known as CCS reads) for individual SMRT Cell ZMWs from high-coverage subreads:

```
$ pbsmrtpipe pipeline-id pbsmrtpipe.pipelines.sa3_ds_ccs \
  -e eid_subread:/data/smrt/2372215/0007/Analysis_Results/\
m150404_101626_42267_c100807920800000001823174110291514_s1_p0.all.subreadset.xml \
  --preset-xml preset.xml -o job_output
```

This pipeline is relatively simple and parallelizes especially well. The essential outputs are a ConsensusRead Data Set (composed of one or more unmapped BAM files) and corresponding FASTA and FASTQ files:

```
job_output/tasks/pbccs.tasks.ccs-0/ccs.consensusreadset.xml
job_output/tasks/pbsmrtpipe.tasks.bam2fasta_ccs-0/file.fasta
job_output/tasks/pbsmrtpipe.tasks.bam2fastq_ccs-0/file.fastq
```

The `pbccs.tasks.ccs-0` task directory will also contain a JSON report with basic metrics for the run such as number of reads passed and rejected for various reasons. (Note, as explained below, that the location of the final ConsensusRead XML - and JSON report - will be different in chunk mode.)

As the full resequencing workflow operates directly on subreads to produce a genomic consensus, it is not applicable to CCS reads. However, a CCS pipeline is available that incorporates the `blasr` mapping step:

```
$ pbsmrtpipe pipeline-id pbsmrtpipe.pipelines.sa3_ds_ccs_align \
  -e eid_subread:/data/smrt/2372215/0007/Analysis_Results/ \
m150404_101626_42267_c100807920800000001823174110291514_s1_p0.all.subreadset.xml \
  -e eid ref dataset:/data/references/lambda.referenceset.xml \
  --preset-xml preset.xml -o job_output
```

Example - Iso-Seq® Transcriptome Analysis

The Iso-Seq Transcriptome Analysis workflows automate use of the `pbtranscript` package for investigating mRNA transcript isoforms. The transcript analysis uses CCS reads where possible, and the pipeline incorporates the CCS pipeline with looser settings. The starting point is therefore still a SubreadSet. The simpler of the two pipelines is `sa3_ds_isoseq_classify`, which runs CCS and classifies the reads as full-length or not:

```
$ pbsmrtpipe pipeline-id pbsmrtpipe.pipelines.sa3_ds_isoseq_classify \
  -e eid_subread:/data/smrt/2372215/0007/Analysis_Results/
m150404_101626_42267_c100807920800000001823174110291514_s1_p0.all.subreadset.xml \
  --preset-xml preset.xml -o job_output
```

The output files from the CCS pipeline will again be present. Note however that the sequences will be lower-quality as the pipeline tries to use as many reads as possible. The output task folder `pbtranscript.tasks.classify-0` (or gathered equivalent; see below) contains the classified transcripts in various ContigSet Data Sets (or underlying FASTA files).

A more thorough analysis yielding Quiver/Arrow-polished, high-quality isoforms is the `pbsmrtpipe.pipelines.sa3_ds_isoseq` pipeline, which is invoked identically to the classify-only pipeline. Note that this is significantly slower, as the clustering step may take days to run for large Data Sets.

Example - Exporting Subreads to FASTA/FASTQ

Converting a PacBio SubreadSet to FASTA or FASTQ format for use with external software can be performed‘ as a standalone pipeline. Unlike most of the other pipelines, this one has no task-specific options and no chunking, so the invocation is very simple:

```
$ pbsmrtpipe pipeline-id pbsmrtpipe.pipelines.sa3_ds_subreads_to_fastx \
  -e eid_subread:/data/smrt/2372215/0007/Analysis_Results/
m150404_101626_42267_c100807920800000001823174110291514_s1_p0.all.subreadset.xml \
  -o job_output
```

The result files will be here:

```
job_output/tasks/pbsmrtpipe.tasks.bam2fasta-0/file.fasta
job_output/tasks/pbsmrtpipe.tasks.bam2fastq-0/file.fastq
```

Both are also available gzipped in the same directories.

Chunking

To take advantage of `pbsmrtpipe’s` parallelization, we need an XML configuration file for global `pbsmrtpipe` options, which can be generated by the following command:

```
$ pbsmrtpipe show-workflow-options -o preset.xml
```

The output `preset.xml` will have this format:

```
<?xml version="1.0" encoding="utf-8" ?>
<pipeline-preset-template>
    <options>
        <option id="pbsmrtpipe.options.max_nproc">
            <value>16</value>
        </option>
        <option id="pbsmrtpipe.options.chunk_mode">
            <value>False</value>
        </option>
        <!-- MANY MORE OPTIONS OMITTED -->
    </options>
</pipeline-preset-template>
```

The appropriate types should be clear; quotes are unnecessary, and boolean values should have initial capitals (`True`, `False`). To enable chunk mode, change the value of option `pbsmrtpipe.options.chunk_mode` to `True`. Several additional options may also need to be modified:

- `pbsmrtpipe.options.distributed_mode` enables execution of most tasks on a managed cluster such as Sun Grid Engine. Use this for chunk mode if available.
- `pbsmrtpipe.options.max_nchunks` sets the upper limit on the number of jobs per task in chunked mode. Note that more chunks is not always better, as there is some overhead to chunking, especially in distributed mode.

- `pbsmrtpipe.options.max_nproc` sets the upper limit on the number of processors per job (including individual chunk jobs). This should be set to a value appropriate for your compute environment.

You can adjust `max_nproc` and `max_nchunks` in the `preset.xml` to consume as many queue slots as you desire, but note that the number of slots consumed will be the product of the two numbers. For some shorter jobs (typically with low-volume input data), it may make more sense to run the job unchunked but still distribute tasks to the cluster (where they will still use multiple cores if allowed).

Once you are satisfied with the settings, add it to your command like this:

```
$ pbsmrtpipe pipeline-id pbsmrtpipe.pipelines.sa3_ds_resequencing \
  --preset-xml preset.xml \
  -e eid_subread:/data/smrt/2372215/0007/Analysis_Results/
m150404_101626_42267_c100807920800000001823174110291514_s1_p0.all.subreadset.xml \
  -e eid_ref_dataset:/data/references/lambda.referenceset.xml
```

Alternately, the options `--force-chunk-mode, --force-distributed`, `--disable-chunk-mode`, and `--local-only` can be used to toggle the chunk/distributed mode settings on the command line; but this will **not** affect the values of `max_nproc` or `max_nchunks`.

If the pipeline runs correctly, you should see an expansion of task folders. The final results for certain steps (alignment, variantCaller, and so on), should end up in the appropriate “gather” directory. For instance, the final gathered FASTA file from quiver should be in `pbsmrtpipe.tasks.gather_contigset-1`. Note that for many Data Set types, the gathered Data Set XML file will often encapsulate multiple BAM files in multiple directories.

HdfSubreadSet to SubreadSet Conversion

If you have existing `bax.h5` files to process with `pbsmrtpipe`, you need to convert them to a SubreadSet before continuing. Bare `bax.h5` files are **not** directly compatible with `pbsmrtpipe`, but an HdfSubreadSet XML file can be easily generated from a `fofn` (file-of-file-names) or folder of `bax.h5` files using the `dataset` tool. (See “dataset” on page 17.)

From a `fofn`, `allTheBaxFiles.fofn`:

```
$ dataset create --type HdfSubreadSet allTheBaxFiles.hdfsubreadset.xml
allTheBaxFiles.fofn
```

Or a directory with all the bax files:

```
$ dataset create --type HdfSubreadSet allTheBaxFiles.hdfsubreadset.xml allTheBaxFiles/
*.bax.h5
```

This can be used as an entry point to the conversion pipeline. PacBio recommends using chunked mode if there is more than one `bax.h5` file, so include the appropriate `preset.xml`:

```
$ pbsmrtpipe pipeline-id pbsmrtpipe.pipelines.sa3_hdfsubread_to_subread \
  --preset-xml preset.xml -e eid_hdfsubread:allTheBaxFiles.hdfsubreadset.xml
```

And use the gathered output XML file as an entry point to the resequencing pipeline from earlier:

```
$ pbsmrtpipe pipeline-id pbsmrtpipe.pipelines.sa3_ds_resequencing \
  --preset-xml preset.xml \
  -e eid_subread:tasks/pbsmrtpipe.tasks.gather_subreadset-0/gathered.xml \
  -e eid_ref_dataset:/data/references/lambda.referenceset.xml
```

Working with Data Sets

Data Sets can also be created for one or more existing `subreads.bam` files or `alignedsubreads.bam` files for use with the pipeline:

```
$ dataset create --type SubreadSet allTheSubreads.subreadset.xml \
  mySubreadBams/*.bam
```

or:

```
$ dataset create --type AlignmentSet allTheMappedSubreads.alignmentset.xml \
  myMappedSubreadBams/*.bam
```

Make sure that all `.bam` files have corresponding `.bai` and `.pbi` index files before generating the Data Set, as these make some operations significantly faster and are required by many programs. You can create indices using `samtools` and `pbindex`, both included in the distribution:

```
$ samtools index subreads.bam
$ pbindex subreads.bam
```

In addition to the BAM-based Data Sets, and HdfSubreadSet, `pbsmrtpipe` also works with two Data Set types based on FASTA format: `ContigSet` (used for both *de novo* assemblies and other collections of contiguous sequences such as transcripts in the Iso-Seq workflows) and `ReferenceSet` (a reference genome). These are created in the same way as BAM Data Sets:

```
$ dataset create --type ReferenceSet
human_genome.referenceset.xml \
genome/chr*.fasta
```

FASTA files can also be indexed for increased speed using `samtools`, and this is again recommended before creating the Data Set:

```
$ samtools faidx chr1.fasta
```

Note that PacBio's specifications for BAM and FASTA files impose additional restrictions on content and formatting; files produce by non-PacBio software are **not** guaranteed to work as input. Use the `pbvalidate` tool to check for format compliance. (See "pbvalidate" on page 82 for details.)

Job Directory Structure

Following are details about the job directory structure, and examples of the files included.

`root-job-dir/`

- `stdout` (`pbsmrtpipe exe` standard out. Includes minimal status updates.)
- `stderr` (`pbsmrtpipe exe` standard err. This should be the first place to look for workflow-level traceback and task errors.)
- `logs/`
  - `pbsmrtpipe.log`
  - `master.log`
- `workflow/`
  - `entry_points.json` (This is essentially a heterogeneous dataset of the `input.xml` or command-line entry points with `entry_id`.)
  - `datastore.json` (This is a fundamental store of all output files, and also contains initial task and workflow level values.)
- `html/`
  - `css/`
  - `js/`
  - `index.html` (The main summary page.)
- `tasks/#` (All tasks are here.)
  - `{task_id}-{instance_id}/#` (Each task has its own directory.)
    `tool-contract.json` (Tool Contract for `{task_id}`)
    `resolved-tool-contract.json` (Resolved Tool Contract; contains paths to files and specific options used in the task execution.)
    `runnable-task.json`
    `task-report.json` (Generated when the task is completed.)
    `outfile.1.txt` (Output files)
    `stdout`
    `stderr`
    `task.log` (This is optional if the task uses `$logfile` in resources.)
    `cluster.stdout` (If this is a non-local job.)
    `cluster.stderr` (If this is a non-local job.)
    `cluster.sh` (`qsub` submission script)
- `{task_id}-{instance_id}/#`(Additional tasks.)

Example datastore.json File

The file paths are in the `workflow/` directory.


```
{
    "files": [
        {
            "file_id": "global-graph-file-node-id",
            "type_id": "PacBio.FileTypes.JsonReport",
            "path": "relative/path/to/report.json",
            "produced_by": "task_instance_id"
        },
        {
            "file_id": "my_file_id",
            "type_id": "PacBio.FileTypes.csv",
            "path": "relative/path/to/f.csv",
            "produced_by": "task_id-instance_id"
        }
    ],
    "version": "0.4.3"
}
```

Task-Specific Components

The are several structured JSON files within a specific task directory:

- `tool-contract.json`
- `resolved-tool-contract.json`
- `runnable-task.json`
- `task-report.json`

Example Tool Contract JSON File


```
{
    "driver": {
        "env": {},
        "exe": "python -m pbsmrtpipe.pb_tasks.dev  run-rtc  ",
        "serialization": "json"
    },
    "tool_contract": {
        "_comment": "Created by v0.4.11",
        "description": "Quick tool dev_reference_ds_report
pbsmrtpipe.tasks.dev_reference_ds_report",
        "input_types": [
            {
                "description": "description for PacBio.DataSet.ReferenceSet_0",
                "file_type_id": "PacBio.DataSet.ReferenceSet",
                "id": "Label PacBio.DataSet.ReferenceSet_0",
                "title": "<DataSetFileType id=PacBio.DataSet.ReferenceSet name=file >"
            }
        ],
        "is_distributed": false,
        "name": "Tool dev_reference_ds_report",
        "nproc": 3,
        "output_types": [
            {
                "default_name": "report",
                "description": "description for <FileType id=PacBio.FileTypes.JsonReport
name=report >",
                "file_type_id": "PacBio.FileTypes.JsonReport",
                "id": "Label PacBio.FileTypes.JsonReport_0",
                "title": "<FileType id=PacBio.FileTypes.JsonReport name=report >"
            }
        ],
        "resource_types": [],
        "schema_options": [
            {
                "$schema": "http://json-schema.org/draft-04/schema#",
                "pb_option": {
                    "default": false,
                    "description": "Option dev_diagnostic_strict description",
                    "name": "Option dev_diagnostic_strict",
                    "option_id": "pbsmrtpipe.task_options.dev_diagnostic_strict",
                    "type": "boolean"
                },
                "properties": {
                    "pbsmrtpipe.task_options.dev_diagnostic_strict": {
                        "default": false,
                        "description": "Option dev_diagnostic_strict description",
                        "title": "Option dev_diagnostic_strict",
                        "type": "boolean"
                    }
                },
                "required": [
                    "pbsmrtpipe.task_options.dev_diagnostic_strict"
```

```
                ],
                "title": "JSON Schema for pbsmrtpipe.task_options.dev_diagnostic_strict",
                "type": "object"
            }
        ],
        "task_type": "pbsmrtpipe.task_types.standard",
        "tool_contract_id": "pbsmrtpipe.tasks.dev_reference_ds_report"
    },
    "tool_contract_id": "pbsmrtpipe.tasks.dev_reference_ds_report",
    "version": "0.1.0"
}
```

Example Resolved Tool Contracts JSON File

```
{
    "driver": {
        "env": {},
        "exe": "python -m pbsmrtpipe.pb_tasks.dev  run-rtc  ",
        "serialization": "json"
    },
    "resolved_tool_contract": {
        "_comment": "Created by pbcommand v0.4.11",
        "input_files": [
            "/Users/mkocher/gh_projects/pbsmrtpipe/testkit-data/dev_diagnostic/referenceset.xml"
        ],
        "is_distributed": false,
        "log_level": "INFO",
        "nproc": 3,
        "options": {
            "pbsmrtpipe.task_options.dev_diagnostic_strict": false
        },
        "output_files": [
            "/Users/mkocher/gh_projects/pbsmrtpipe/testkit-data/dev_diagnostic/job_output/tasks/
pbsmrtpipe.tasks.dev_reference_ds_report-0/report.json"
        ],
        "resources": [],
        "task_type": "pbsmrtpipe.task_types.standard",
        "tool_contract_id": "pbsmrtpipe.tasks.dev_reference_ds_report"
    }
}
```

Example Runnable Task JSON File

In each task directory, a runnable-task is written. It contains all the metadata necessary for the task to be run on the execution node, or run locally. The task manifest is run using the `pbtools-runner` command line tool.

There are several reasons for the `pbtool-runner` abstraction:

- NFS checks to validate input files can be found. (This is related to python NFS caching errors that often result in IOErrors.)
- Create and clean up temporary resources on the execution node.
- Write `env.json` files to document the environment variables.
- Write metadata results about the output of the tasks.
- Allow some tweaking and rerunning by hand of failed tasks.
- Strict documenting of input files, resolved task type, and resolved task options used to run the task.

`runnable-task.json` contains all the resolved task types and resolved task values, such as resolved options, input files, output files, and resources (such as temporary files and temporary directories).


```
{
    "cluster": {
        "start": "qsub -S /bin/bash -sync y -V -q default -N ${JOB_ID} \\\n    -o
\"${STDOUT_FILE}\" \\\n    -e \"${STDERR_FILE}\" \\\n    -pe smp ${NPROC} \\\n    \"${CMD}\"",
        "stop": "qdel ${JOB_ID}"
    },
    "env": {},
    "id": "pbsmrtpipe.tasks.dev_reference_ds_report",
    "resource_types": [],
    "task": {
        "cmds": [
            "python -m pbsmrtpipe.pb_tasks.dev  run-rtc   /Users/mkocher/gh_projects/pbsmrtpipe/
testkit-data/dev_diagnostic/job_output/tasks/pbsmrtpipe.tasks.dev_reference_ds_report-0/resolved-
tool-contract.json"
        ],
        "input_files": [
            "/Users/mkocher/gh_projects/pbsmrtpipe/testkit-data/dev_diagnostic/referenceset.xml"
        ],
        "is_distributed": false,
        "nproc": 3,
        "options": {
            "pbsmrtpipe.task_options.dev_diagnostic_strict": false
        },
        "output_dir": "/Users/mkocher/gh_projects/pbsmrtpipe/testkit-data/dev_diagnostic/
job_output/tasks/pbsmrtpipe.tasks.dev_reference_ds_report-0",
        "output_files": [
            "/Users/mkocher/gh_projects/pbsmrtpipe/testkit-data/dev_diagnostic/job_output/tasks/
pbsmrtpipe.tasks.dev_reference_ds_report-0/report.json"
        ],
        "resources": [],
        "task_id": "pbsmrtpipe.tasks.dev_reference_ds_report",
        "task_type_id": "pbsmrtpipe.tasks.dev_reference_ds_report",
        "uuid": "252d72cd-4617-4a33-9622-606720dec512"
    },
    "version": "0.44.2"
}
```


Example task-report.json

After `pbtool-runner` completes executing the task, a metadata job report is written to the job directory. `task-report.json` contains basic metadata about the completed task.


```
{
    "_changelist": 127707,
    "_version": "2.1",
    "attributes": [
        {
            "id": "workflow_task.host",
            "name": null,
            "value": "mp-f027.nanofluidics.com"
        },
        {
            "id": "workflow_task.run_time",
            "name": null,
            "value": 3563
        },
        {
```

```
            "id": "workflow_task.exit_code",
            "name": null,
            "value": 0
        },
        {
            "id": "workflow_task.error_msg",
            "name": null,
            "value": ""
        }
    ],
    "id": "workflow_task",
    "plotGroups": [],
    "tables": []
}
```

pbsv

`pbsv` is a structural variant caller for PacBio reads. It identifies large (default: ≥50 bp) insertions and deletions in a sample or set of samples relative to a reference genome. `pbsv` takes as input PacBio reads (BAM) and a reference genome (FASTA); it outputs structural variant calls (VCF and BED).

Usage:

```
pbsv [-h] [--version] [--quiet] [--verbose]
            {align,call,generate-config} ...
```

Options	Description
`-h, --help`	Displays help information and exits.
`--version`	Displays program version number and exits.
`--quiet`	Runs silently. (Default = `False`)
`--verbose`	Sets the verbosity level. (Default = `False`)
`generate-config`	Generate a configuration file for the `pbsv` tool suite.
`align`	Align reads in a BAM file to a reference genome, chain alignments, mark duplicates, then sort.
`call`	Call structural variants from long read alignments.

pbsv generate-config

This command is **optional**, and is used to produce a template configuration file initialized with defaults for all parameters for downstream stages. It is **only** needed to specify non-default configurations.

`pbsv` does **not** accept optional parameters at the command line. Instead, it accepts a configuration file that specifies all optional parameters. The `generate-config` command creates a configuration file with all parameters set to the defaults. This file can be passed as `--cfg_fn=out.cfg` to all other `pbsv` commands.

- Sections of the configuration file are specified with square brackets. `[call]` marks the start of settings for the `pbsv call` command.
- Settings are specified as `param=value`, such as `svlength=50`.

- Lines that start with # are comments.

Usage:

```
pbsv generate-config [-h] [-o sv.cfg]
```

Options	Description
`-h, --help`	Displays help information and exits.
`-o sv.cfg, --cfg_fn sv.cfg`	A `pbsv` configuration file. If `-o` is **not** specified, then the configuration file is written to `stdout`.

pbsv align

`pbsv align` aligns reads to a reference genome. Input reads are accepted in the following formats: `subreads.bam` (preferred); plain text, gzipped or bzipped FASTA/Q; or `subreadset.xml`.

Initial local alignments are generated with NGM-LR (`pbsvutil ngmlr`); co-linear alignments from a read are then chained (`pbsvutil chain`); duplicate subreads are flagged (`pbsvutil markduplicates`); and the alignments are sorted by chromosome position. Sample information is output as read group annotation in the aligned BAM.

Usage:

```
pbsv align [-h] [--cfg_fn sv.cfg] [--movienames2samples_json
m2s.json] ref_fn reads_bam align_bam
```

Required	Description
`ref_fn`	Reference genome in FASTA or ReferenceSet format. (`.fai` index and NGM-LR index are required.) The reference genome is produced by `fasta-to-reference`.
`reads_bam`	Input raw reads.
`align_bam`	Output BAM/SAM file sorted by coordinate.

Options	Description
`-h, --help`	Displays help information and exits.
`--cfg_fn sv.cfg`	A `pbsv` configuration file.
`--movienames2samples_json m2s.json`	A JSON file with a map from movie name to sample name, used to populate the sample name, is read group headers in the output BAM. Format: `[ [“movie1a”, “sample1”], [“movie1b”, “sample1”], … [“movieN”, “sampleN”] ]`

As with all `pbsv` commands, any parameters are specified in the configuration file.

`pbsv align` uses a multistep process. The individual steps are exposed through `pbsvutil`, a companion utility tool. The steps are:

1. `pbsvutil x2fasta` - Extract FASTA reads from multiple input formats: plain text, gzipped or bzipped FASTA and FASTQ; subreads BAM; subread set XML.
2. `pbsvutil ngmlr` - Align FASTA reads to a reference genome with NGM-LR. (**https://github.com/philres/ngmlr/**). The reference genome is produced by `fasta-to-reference`.
3. `pbsvutil chain` - Merge co-linear alignments from a read into a single alignment.
4. `pbsvutil markduplicates` - Flag duplicate subread alignments, selecting only a single subread per ZMW as non-duplicate.
5. Sort by position.

pbsv call

`pbsv call` accepts aligned reads for one or more samples in the style produced by `pbsv align`; it outputs variant calls in BED and VCF file formats (`sv.bed` and `sv.vcf`). The file output names are inferred from the last parameter. Sample information is obtained from the read groups in the aligned reads. Reads with no sample annotation are assigned to “`UnnamedSample`”.

Usage:

```
pbsv call [-h] [--cfg_fn sv.cfg] [--reference_regions
regions] ref_fn alignments out_bed
```

Required	Description
`ref_fn`	Reference genome in FASTA or ReferenceSet format. (`.fai` index and NGM-LR index are required.) The reference genome is produced by `fasta-to-reference`.
`alignments`	Input reads mapped by `pbsv align` in BAM, SAM, FOFN, or AlignmentSet format.
`out_bed`	Output structural variant calls. Two output files are written, one in BED format (`.bed`) and one in VCF format (`.vcf`). The file names are inferred from `out_bed`.

Options	Description
`-h, --help`	Displays help information and exits.
`--cfg_fn sv.cfg`	A `pbsv` configuration file.
`--reference_regions regions`	Limit analysis to a union of specified regions, specified as a semicolon-delimited list of chromosome ranges. If `--reference_regions` is **not** specified, the analysis is run on the entire genome. Accepted region formats are: `chromName:chromStart-chromEnd`; `chromName:chromStart` (which means from `chromStart` to the end of the chromosome); and `chromName`, which means the entire chromosome.

pbtranscript

The `pbtranscript` tool is part of the Iso-Seq® analysis pipeline, and it is used for the Classify and Cluster/polish steps, as well as post-polish analysis.

Using the command-line, the Iso-Seq analysis is performed in 3 steps:

1. Run CCS on your subreads, generating a CCS BAM file. Then generate an XML file from the BAM file.
2. Run Classify on your CCSs with the XML as input, generating a FASTA file of annotated sequences.
3. Run Cluster on the FASTA file produced by Classify, generating polished isoforms.

Step 1: CCS

Convert the subreads to circular consensus sequences, using the following command:

```
ccs --noPolish --minLength=300 --minPasses=1 --minZScore=-999 --maxDropFraction=0.8 --minPredictedAccuracy=0.8 --minSnr=4 subreads.bam ccs.bam
```

Where:

- `ccs.bam` is where the CCSs will be output.
- `subreads.bam` is the file containing your subreads.

If you think that you have transcripts of interest that are less than 300 base pairs in length, be sure to adjust the `minLength` parameter. Next, you generate an XML file from your CCSs, using the following command:

```
dataset create --type ConsensusReadSet ccs.xml ccs.bam
```

Where:

- `ccs.xml` is the name of the XML file you are generating.
- `ccs.bam` is the name of the BAM file you generated previously using the `ccs` command.

Step 2: Classify

Iso-Seq Classify classifies reads into full-length or non-full-length reads, artificial-concatemer chimeric, or non-chimeric reads.

To classify a read as full-length or non-full-length, we search for primers and polyA within reads. If and **only** if both a primer and polyAs are seen in a read, it is classified as a **full-length read**. Otherwise, the read is classified as **non-full-length**. We also remove primers and polyAs from reads and identify read-strandedness based on this information.

Next, full-length reads are classified into artificial-concatemer chimeric reads or non-chimeric reads by locating primer hits within reads.

- HMMER: `phmmer` in the HMMER package is used to detect locations of primer hits within reads and classify reads which have primer hits in the middle of sequences as artificial-concatemer chimeric.

Classify - Input File

- `ccs.xml`: Circular consensus sequences generated from the CCS step.

Classify - Output Files

- `isoseq_flnc.fasta`: Contains all full-length, non-artificial-concatemer reads.
- `isoseq_nfl.fasta`: Contains all non-full-length reads.
- `isoseq_draft.fasta`: An intermediate file in order to get full-length reads, which you can ignore.

Reads in these FASTA output files look like the following:

```
>m140121_100730_42141_c100626750070000001823119808061462_s1_p0/119/30_1067_CCS
strand=+;fiveseen=1;polyAseen=1;threeseen=1;fiveend=30;polyAend=1067;threeend=1096;pri
mer=1;chimera=0
ATAAGACGACGCTATATG
```

These lines have the format:

```
<movie_name>/<ZMW>/<start>_<end>_CCS INFO
```

The `INFO` fields are:

- `strand`: Either + or `-`, whether a read is forward or reverse-complement cDNA.
- `fiveseen`: Whether 5' primer is seen in this read, `1` is yes, `0` is no.
- `polyAseen`: Whether polyA tail is seen, `1` is yes, `0` is no.
- `threeseen`: Whether 3' primer is seen, `1` is yes, `0` is no.
- `fiveend`: Start position of 5' in the read.
- `threeend`: Start position of 3' in the read.
- `polyAend`: Start position of polyA in the read.
- `primer`: Index of primer seen in this read.
- `chimera`: Whether this read is classified as a chimeric cDNA.

- `classify_summary.txt`: This file contains the following statistics:
  - Number of reads of insert
  - Number of 5’ reads
  - Number of 3’ reads
  - Number of polyA reads
  - Number of filtered short reads
  - Number of non-full-length reads
  - Number of full-length reads
  - Number of full-length non-chimeric reads
  - Average full-length non-chimeric read length

Note: By seeing that the number of full-length, non-chimeric (`flnc`) reads is only a little less than the number of full-length reads, we can confirm that the number of artificial concatemers is very low. This indicates a successful SMRTbell library preparation.

Classify - Usage

```
pbtranscript classify [OPTIONS] ccs.xml isoseq_draft.fasta --flnc=isoseq_flnc.fasta --
nfl=isoseq_nfl.fasta
```

- Where `ccs.xml` is the XML file you generated in Step 1.
- `isoseq_flnc.fasta` contains only the full-length, non-chimeric reads.
- `isoseq_nfl.fasta` contains all non-full-length reads.

Or you can run Classify creating XML files instead of FASTA files as follows:

```
pbtranscript classify [OPTIONS] ccs.xml isoseq_draft.fasta --
flnc=isoseq_flnc.contigset.xml --nfl=isoseq_nfl.contigset.xml
```

- Where `ccs.xml` is the XML file you generated in Step 1.
- `isoseq_flnc.contigset.xml` contains only the full-length, non-chimeric reads.
- `isoseq_nfl.contigset.xml` contains all non-full-length reads.

Note: One can always use `pbtranscript subset` to further subset `isoseq_draft.fasta` if `--flnc` and `--nfl` are **not** specified when you run `pbtranscript classify`. For example:

```
pbtranscript subset isoseq_draft.fasta isoseq_flnc.fasta --FL --nonChimeric
```

Classify Options

- To view Classify options, enter `pbtranscript classify --help`.

Required	Description
`readsFN`	First positional argument. Input CCS reads in BAM, Data Set XML, or FASTA format. Example: `ccs.bam,xml,fasta.`
`outReadsFN`	Second positional argument. Output file which contains all classified reads in FASTA or contigset XML format. Example: `isoseq_draft.fasta,contigset.xml`

Options	Description
`--flnc`	Outputs full-length non-chimeric reads in FASTA or contigset XML format. Example: `FLNC_FA.fasta,contigset.xml`
`-d OUTDIR, --outDir OUTDIR`	Directory to store `HMMER` output. (Default = `output/`)
`-summary`	Text file to output classify summary. (Default = `out.classify_summary.txt`).
`-p primers.fa, --primer primers.fa`	Primer FASTA file. (Default = `primers.fa`)

Options	Description
`--report`	CSV file of primer information. Contains the same information found in the description lines of the output FASTA. (Default = `out.primer_info.csv`)
`-cpus CPUS`	Number of CPUs to run `HMMER`. (Default = `8`)
`--min_seq_len MIN_SEQ_LEN`	Minimum CCS length to be analyzed. Fragments shorter than the minimum sequence length are excluded from analysis. (Default = `300`)
`--min_score MIN_SCORE`	Minimum `phmmer` score for primer hit. (Default = `10`)
`--detect_chimera_nfl`	Detect chimeric reads among non-full-length reads. Non-full-length non-chimeric/chimeric reads are saved to `outDir/nflnc.fasta` and `outDir/nflc.fasta`.
`--ignore_polyA`	Full-length criteria does **not** require polyA tails. By default this is off, which means that polyA tails are required for a sequence to be considered full length. When it is turned on, sequences do **not** need polyA tails to be considered full length.

Step 3: Cluster and Polish

Iso-Seq Cluster performs isoform-level clustering using the Iterative Clustering and Error correction (ICE) algorithm, which iteratively classifies full-length non-chimeric CCS reads into clusters and builds consensus sequences of clusters using `pbdagcon`.

ICE is customized to work well on alternative isoforms and alternative polyadenlynation sites, but **not** on SNP analysis and SNP-based highly complex gene families.

Iso-Seq Polish further polishes consensus sequences of clusters (i.e., `pbdagcon` output) taking into account all the QV information. Full-length non-chimeric CCS reads and non-full-length CCS reads are assigned into clusters based on similarity. Then for each cluster, we align raw subreads of its assigned ZMWs towards its consensus sequence. Finally, we load quality values to these alignments and polish the consensus sequence using `quiver` or `Arrow`.

Cluster - Input Files

- A file of non-full length reads output by Classify.
- A file of full-length non-chimeric reads.

Cluster - Output Files

- A file of polished, high-quality consensus sequences.
- A file of polished, low-quality consensus sequences.
- `cluster_summary.txt`, which contains the following statistics:
  - Number of consensus isoforms.
  - Average read length of consensus isoforms.
- `cluster_report.csv`; each line contains the following fields:
  - `cluster_id`: ID of a consensus isoforms from ICE.
  - `read_id`: ID of a read which supports the consensus isoform.

– `read_type`: Type of the supportive read.

Cluster - Usage

```
pbtranscript cluster [OPTIONS] isoseq_flnc.fasta polished_clustered.fasta --quiver --
nfl=isoseq_nfl.fasta --bas_fofn=my.subreadset.xml
```

Or

```
pbtranscript cluster [OPTIONS] isoseq_flnc.contigset.xml
polished_clustered.contigset.xml --quiver --nfl=isoseq_nfl.contigset.xml --
bas_fofn=my.subreadset.xml
```

Note: `--quiver --nfl=isoseq_nfl.fasta|contigset.xml` **must** be specified to get Quiver/Arrow-polished consensus isoforms.

Optionally, you may call the following command to run ICE and create unpolished consensus isoforms **only**:

```
pbtranscript cluster [OPTIONS] isoseq_flnc.fasta unpolished_clustered.fasta
```

Cluster Options

- To view Cluster options, use `pbtranscript cluster --help`.

Options	Description
`Input reads`	Input full-length non-chimeric reads in FASTA or contigset XML format. Used for clustering consensus isoforms. Example: `isoseq_flnc.fasta,contigset.xml` (**Required**)
`Output Isoforms`	Output predicted (unpolished) consensus isoforms in a FASTA file. Example: `out.fasta,congitset.xml` (**Required**)
`--nfl_fa isoseq_nfl.fasta`	Input non-full-length reads in FASTA format, used for polishing consensus isoforms.
`--ccs_fofn ccs.fofn`	A `ccs.fofn`, `ccs.bam` or `ccs.xml` file. If not given, Cluster assumes there is no QV information available.
`--bas_fofn my.subreadset.xml`	A file which provides quality values of raw reads and subreads. Can be either a `fofn` (file-of-file-names) of BAM or BAX files, or a Data Set XML.
`-d output/, --outDir output/`	Directory to store temporary and output cluster files. (Default = `output/`)
`--tmp_dir tmp/`	Directory to store temporary files. (Default = `tmp/`)
`--summary my.cluster_summary.txt`	Text file to output cluster summary. (Default = `my.cluster_summary.txt`)
`--report report.csv`	A CSV file, each line containing a cluster, an associated read of the cluster, and the read type.
`--pickle_fn PICKLE_FN`	Developers' option from which all clusters can be reconstructed.
`--cDNA_size`	Estimated cDNA size. Values = `[under1k, between1k2k, between2k3k, above3k]`
`--quiver`	Call `Quiver` or `Arrow` to polish consensus isoforms using non-full-length non-chimeric CCS reads.
`--use_finer_qv`	Use finer classes of QV information from CCS input instead of a single QV from FASTQ. This option is slower and consumes more memory.
`--use_sge`	Specify that the cluster uses SGE.

Options	Description
`--max_sge_jobs MAX_SGE_JOBS`	The maximum number of jobs that will be submitted to SGE concurrently.
`--unique_id UNIQUE_ID`	Unique ID for submitting SGE jobs.
`--blasr_nproc BLASR_NPROC`	Number of cores for each `blasr` job.
`--quiver_nproc QUIVER_NPROC`	Number of CPUs that each `quiver/Arrow` job uses.
`--hq_quiver_min_accuracy HQ_QUIVER_MIN_ACCURACY`	Minimum allowed quiver accuracy to classify an isoform as hiqh-quality.
`-qv_trim_5 QV_TRIM_5`	Ignore QV of n bases in the 5' end.
`--qv_trim_3 QV_TRIM_3`	Ignore QV of n bases in the 3' end.
`--hq_isoforms_fa output/all_quivered_hq.fa`	Quiver/Arrow-polished, high-quality isoforms in FASTA format. (Default = `output/all_quivered_hq.fa`)
`--hq_isoforms_fq output/all_quivered_hq.fq`	Quiver/Arrow-polished, high-quality isoforms in FASTQ format. (Default = `output/all_quivered_hq.fq`)
`--lq_isoforms_fa output/all_quivered_lq.fa`	Quiver/Arrow-polished, low-quality isoforms in FASTA format. (Default = `output/all_quivered_lq.fa`)
`--lq_isoforms_fq output/all_quivered_lq.fq`	Quiver/Arrow-polished, low-quality isoforms in FASTQ format. (Default = `output/all_quivered_lq.fq`)

Subset Options

`Subset` is an optional program which can be used to subset the output files for particular classes of sequences, such as non-chimeric reads, or non-full-length reads.

- To view Subset options, enter `pbtranscript subset --help`.

Options	Description
`Input sequences`	Input FASTA file. Example: `isoseq_draft.fasta` (**Required**)
`Output sequences`	Output FASTA file. Example: `isoseq_subset.fasta` (**Required**)
`--FL`	Output only full-length reads, with 3' primer and 5' primer and polyA tail seen.
`--nonFL`	Output only non-full-length reads.
`--nonChimeric`	Output only non-chimeric reads.
`--printReadLengthOnly`	Only print read lengths, with no read names and sequences.
`--ignore_polyA`	Full-Length criteria does **not** require polyA tails. By default this is off, which means that polyA tails are required for a sequence to be considered full length. When it is turned on, sequences do **not** need polyA tails to be considered full length.

pbvalidate

The `pbvalidate` tool validates that files produced by PacBio software are compliant with Pacific Biosciences' own internal specifications.

Input Files

`pbvalidate` supports the following input formats:

- BAM

- FASTA
- Data Set XML

See **http://pacbiofileformats.readthedocs.org/en/3.0/** for further information about each format's requirements.

Usage

```
pbvalidate [-h] [--version] [--log-file LOG_FILE]
           [--log-level {DEBUG,INFO,WARNING,ERROR,CRITICAL} | --debug | --quiet | -v]
           [-c] [--quick] [--max MAX_ERRORS]
           [--max-records MAX_RECORDS]
           [--type
{BAM,Fasta,AlignmentSet,ConsensusSet,ConsensusAlignmentSet,SubreadSet,BarcodeSet,Conti
gSet,ReferenceSet,GmapReferenceSet,HdfSubreadSet}]
           [--index] [--strict] [-x XUNIT_OUT] [--unaligned]
           [--unmapped] [--aligned] [--mapped]
           [--contents {SUBREAD,CCS}] [--reference REFERENCE]
           file
```

Required	Description
`file`	BAM, FASTA, or Data Set XML file to validate.

Options	Description
`-h, --help`	Displays help information and exits.
`--version`	Displays program version number and exits.
`--log-file LOG_FILE`	Write the log to file. Default (`None`) will write to `stdout`.
`--log-level`	Specify the log level; values are `[DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL.]` (Default = `CRITICAL`)
`--debug=False`	Alias for setting the log level to `DEBUG`. (Default = `False`)
`--quiet`	Alias for setting the log level to `CRITICAL` to suppress output. (Default = `False`)
`--verbose, -v`	Sets the verbosity level. (Default = `None`)
`--quick`	Limits validation to the first 100 records (plus file header); equivalent to `--max-records=100.` (Default = `False`)
`--max MAX_ERRORS`	Exit after `MAX_ERRORS` have been recorded. (Default = `None`; check the entire file.)
`--max-records MAX_RECORDS`	Exit after `MAX_RECORDS` have been inspected. (Default = `None`; check the entire file.)
`--type`	Use the specified file type instead of guessing. `[BAM,Fasta,AlignmentSet,ConsensusSet,ConsensusAlignmentSet,SubreadSet,BarcodeSet,ContigSet,ReferenceSet,GmapReferenceSet,HdfSubreadSet]` (Default = `None`)
`--index`	Require index files:`.fai` or `.pbi`. (Default = `False`)
`--strict`	Turn on additional validation, primarily for Data Set XML. (Default = `False`)

BAM Options	Description
`--unaligned`	Specify that the file should contain **only** unmapped alignments. (Default = `None`, no requirement.)
`--unmapped`	Alias for `--unaligned`. (Default = `None`)
`--aligned`	Specify that the file should contain **only** mapped alignments. (Default = `None`, no requirement.)
`--mapped`	Alias for `--aligned`. (Default = `None`)
`--contents`	Enforce the read type: `[SUBREAD, CCS]` (Default = `None`)
`--reference REFERENCE`	Path to optional reference FASTA file, used for additional validation of mapped BAM records. (Default = `None`)

Examples

To validate a BAM file:

```
$ pbvalidate in.subreads.bam
```

To validate a FASTA file:

```
$ pbvalidate in.fasta
```

To validate a Data Set XML file:

```
$ pbvalidate in.subreadset.xml
```

To validate a BAM file and its index file (`.pbi`):

```
$ pbvalidate --index in.subreads.bam
```

To validate a BAM file and exit after 10 errors are detected:

```
$ pbvalidate --max 10 in.subreads.bam
```

To validate up to 100 records in a BAM file:

```
$ pbvalidate --max-records 100 in.subreads.bam
```

To validate up to 100 records in a BAM file (equivalent to `--max-records=100`):

```
$ pbvalidate --quick in.subreads.bam
```

To validate a BAM file, using a specified log level:

```
$ pbvalidate --log-level=INFO in.subreads.bam
```

To validate a BAM file and write log messages to a file rather than to `stdout`:

```
$ pbvalidate --log-file validation_results.log in.subreads.bam
```

quiver

This is the `variantCaller` tool with the consensus algorithm set to `quiver`. **Note**: `quiver` operates on PacBio RS II data **only**. See “variantCaller” on page 86 for details.

sawriter

The `sawriter` tool generates a suffix array file from an input FASTA file. It is used to prebuild suffix array files for reference sequences which can later be used in resequencing workflows. `sawriter` comes with `blasr`, and is independent of `python`.

Usage

```
sawriter saOut fastaIn [fastaIn2 fastaIn3 ...] [-blt p] [-larsson] [-4bit] [-manmy]
[-kar]
   or
sawriter fastaIn  (writes to fastIn.sa)
```

Options	Description
`-blt p`	Build a lookup table on prefixes of length p. This speeds up lookups considerably (more than the LCP table), but misses matches less than p when searching.
`-4bit`	Read in one FASTA file as a compressed sequence file.
`-larsson`	Uses the Larsson and Sadakane method to build the array. (Default)
`-mamy`	Uses the MAnber and MYers method to build the array. This is slower than the Larsson method, and produces the same result. This is mainly for double-checking the correctness of the Larsson method.
`-kark`	Uses the Karkkainen DS3 method for building the suffix array. This is probably slower than the Larsson method, but takes only `N/(sqrt 3)`extra space.
`-welter`	Use lightweight suffix array construction. This is a bit slower than the normal Larsson method.
`-welterweight N`	Use a difference cover of size N for building the suffix array. Valid values are `7`, `32`, `64`, `111`, and `2281`.

summarize Modifications

The `summarizeModifications` tool generates a GFF summary file (`alignment_summary.gff`) from the output of base modification analysis (i.e. `ipdSummary`) combined with the coverage summary GFF generated by resequencing pipelines. This is also part of the standard Base Modification pipelines in `pbsmrtpipe`, and is useful for power users running custom workflows.

Usage

```
summarizeModifications [-h] [--version] [--emit-tool-contract]
                              [--resolved-tool-contract RESOLVED_TOOL_CONTRACT]
                              [--log-file LOG_FILE]
                              [--log-level {DEBUG,INFO,WARNING,ERROR,CRITICAL} | --debug
                              | --quiet | -v]
                              modifications alignmentSummary gff_out
```

Input Files

- `modifications`: Base Modification GFF file.

- alignmentSummary: Alignment Summary GFF file.

Output Files

- gff_out: Coverage summary for regions (bins) spanning the reference with Base Modification results for each region.

Options	Description
-h, --help	Displays help information and exits.
--version	Displays program version number and exits.
--emit-tool-contract	Outputs the tool contract to stdout. (Default = False)
--resolved-tool-contract RESOLVED_TOOL_CONTRACT	Run the tool directly from a PacBio Resolved tool contract. (Default = None)
--log-file LOG_FILE	Write the log to file. Default (None) will write to stdout.
--log-level	Specify the log level; values are [DEBUG, INFO, WARNING, ERROR, CRITICAL] (Default = INFO)
--debug	Alias for setting the log level to DEBUG. (Default = False)
--quiet	Alias for setting the log level to CRITICAL to suppress output. (Default = False)
--verbose, -v	Sets the verbosity level. (Default = None)

variantCaller

variantCaller is a variant-calling tool provided by the GenomicConsensus package which provides several variant-calling algorithms for PacBio sequencing data.

Usage

```
variantCaller    -j8 --algorithm=arrow \
                 -r lambdaNEB.fa        \
                 -o variants.gff        \
                 aligned_subreads.bam
```

This example requests variant-calling, using 8 worker processes and the Arrow algorithm, taking input from the file aligned_subreads.bam, using the FASTA file lambdaNEB.fa as the reference, and writing output to variants.gff.

A particularly useful option is --referenceWindow/-w; which allows the variant-calling to be performed exclusively on a **window** of the reference genome.

Input Files

- A sorted file of reference-aligned reads in Pacific Biosciences' standard BAM format.
- A FASTA file that follows Pacific Biosciences' FASTA file convention.

Note: The quiver and arrow algorithms require that certain metrics are in place in the input BAM file.

- `quiver`, which operates on PacBio RS II data **only**, requires the basecaller-computed "pulse features" `InsertionQV`, `SubstitutionQV`, `DeletionQV`, and `DeletionTag`. These features are populated in BAM tags by the `bax2bam` conversion program. See “bax2bam” on page 5 for details.
- `arrow`, which operates on PacBio RS II P6-C4 data and all Sequel data, requires per-read SNR metrics, and the per-base `PulseWidth` metric for Sequel data (but **not** for PacBio RS II P6-C4). These metrics are populated by Sequel instrument software or the `bax2bam` converter (for PacBio RS II data).

The selected algorithm will stop with an error message if any features that it requires are unavailable.

Output Files

Output files are specified as arguments to the `-o` flag. The file name extension provided to the `-o` flag is meaningful, as it determines the output file format. For example:

```
variantCaller aligned_subreads.bam -r lambda.fa  -o myVariants.gff -o myConsensus.fasta
```

will read input from `aligned_subreads.bam`, using the reference `lambda.fa`, and send variant call output to the file `myVariants.gff`, and consensus output to `myConsensus.fasta`.

The file formats presently supported, by extension, are:

- `.gff`: PacBio GFFv3 variants format; convertable to VCF or BED.
- `.fasta`: FASTA file recording the consensus sequence calculated for each reference contig.
- `.fastq`: FASTQ file recording the consensus sequence calculated for each reference contig, as well as per-base confidence scores

Options	Description
`-j`	The number of worker processes to use.
`--algorithm=`	The variant-calling algorithm to use; values are `plurality`, `quiver`, and `arrow`.
`-r`	The FASTA reference file to use.
`-o`	The output file format; values are `.gff`, `.fasta`, and `.fastq`.
`--maskRadius`	When using the `arrow` algorithm, setting this parameter to a value N greater than 0 will cause `variantCaller` to pass over the data a second time after masking out regions of reads that have >70% errors in 2*N+1 bases. This setting has little to no effect at low coverage, but for high-coverage datasets (>50X), setting this parameter to 3 may improve final consensus accuracy. In rare circumstances, such as misassembly or mapping to the wrong reference, enabling this parameter **may** cause worse performance.

Options	Description
`--minConfidence MINCONFIDENCE` `-q MINCONFIDENCE`	The minimum confidence for a variant call to be output to variants.{gff,vcf} (Default = `40`)
`--minCoverage MINCOVERAGE` `-x MINCOVERAGE`	The minimum site coverage that must be achieved for variant calls and consensus to be calculated for a site. (Default = `5`)

Available Algorithms

At this time there are three algorithms available for variant calling: `plurality`, `quiver`, and `arrow`.

- `plurality` is a simple and very fast procedure that merely tallies the most frequent read base or bases found in alignment with each reference base, and reports deviations from the reference as potential variants. This is a very insensitive and flawed approach for PacBio sequence data, which is prone to insertion and deletion errors.
- `quiver` is a more complex procedure based on algorithms originally developed for CCS. `Quiver` leverages the quality values (QVs) provided by upstream processing tools, which provide insight into whether insertions/deletions/substitutions were deemed likely at a given read position. Use of `quiver` requires the ConsensusCore library. **Note**: `quiver` operates on PacBio RS II data **only**.
- `arrow` is the successor to `quiver`; it uses a more principled HMM model approach. It does **not** require basecaller quality value metrics; rather, it uses the per-read SNR metric and the per-pulse `pulsewidth` metric as part of its likelihood model. Beyond the model specifics, other aspects of the Arrow algorithm are similar to `quiver`. Use of `arrow` requires the ConsensusCore2 library, which is provided by the unanimity codebase.

Confidence Values

The `arrow`, `quiver`, and `plurality` algorithms make a confidence metric available for every position of the consensus sequence. The confidence should be interpreted as a phred-transformed posterior probability that the consensus call is incorrect; such as:

$$QV = -10\log_{10}(p_{err})$$

`variantCaller` clips reported QV values at 93; larger values **cannot** be encoded in a standard FASTQ file.

Chemistry Specificity

The `quiver` and `arrow` algorithm parameters are trained per-chemistry. `quiver` and `arrow` identify the sequencing chemistry used for each run by looking at metadata contained in the data file (the input BAM or `cmp.h5` file). This behavior can be overriden by a command-line option.

When multiple chemistries are represented in the reads in the input file, Quiver/Arrow will model reads appropriately using the parameter set for its chemistry, thus yielding optimal results.

Third Party Command-Line Tools

Following is information on the third-party command-line tools included in the `smrtcmds/bin` subdirectory.

bamtools

- A C++ API and toolkit for reading, writing, and manipulating BAM files.
- See **https://sourceforge.net/projects/bamtools/** for details.

daligner, LAsort, LAmerge, HPC.daligner

- Finds all significant local alignments between reads.
- See **https://dazzlerblog.wordpress.com/command-guides/daligner-command-reference-guide/** for details.

datander

- Finds all local self-alignment between long, noisy DNA reads.
- See **https://github.com/thegenemyers/DAMASKER** for details.

DB2fasta, DBdump, DBdust, DBrm, DBshow, DBsplit, DBstats, Fasta2DB

Utilities that work with Dazzler databases:

- `DB2fasta`: Converts database files to FASTS format.
- `DBdust`: Runs the DUST algorithm over the reads in the untrimmed database, producing a track that marks all intervals of low complexity sequence.
- `DBdump/DBshow`: Displays a subset of the reads in the database; selects the information to show about the reads, including any mask tracks.
- `DBrm:` Deletes all the files in a given database.
- `DBsplit`: Divides a database conceptually into a series of blocks.
- `DBstats`: Shows overview statistics for all the reads in the trimmed database.
- `Fasta2DB`: Builds an initial database, or adds to an existing database, using a list of `.fasta` files.
- See **https://dazzlerblog.wordpress.com/command-guides/dazz_db-command-guide/** for details.

gmap, gmap_build, gmapl

- A genomic mapping and alignment program for mRNA and EST Sequences.
- See **http://research-pub.gene.com/gmap/** for details.

ipython

- An interactive shell for using the Pacific Biosciences API.
- See **https://ipython.org/** for details.

python

- An object-oriented programming language.
- See **https://www.python.org/** for details.

REPmask, TANmask, HPC.REPmask, HPC.TANmask

- A set of programs to soft-mask all tandem and interspersed repeats in Dazzler databases when computing overlaps.
- See **https://github.com/thegenemyers/DAMASKER** for details.

samtools

- A set of programs for interacting with high-throughput sequencing data in SAM/BAM/VCF formats.
- See **http://www.htslib.org/** for details.

P/N 100-939-900 Version 03, Feb 2018