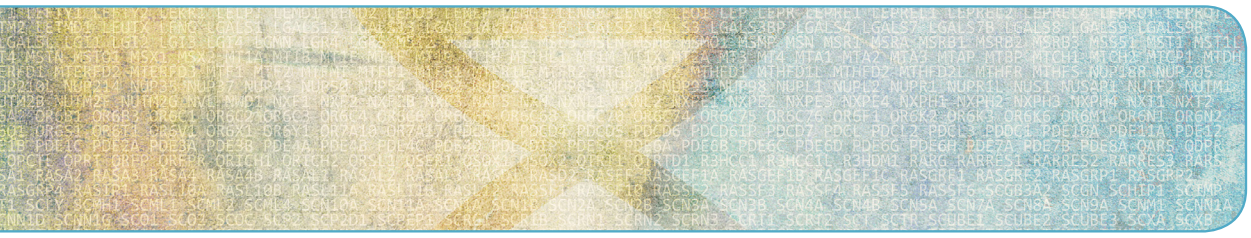
xGen® Duplex Seq Adapters—Tech Access:

Processing sequence data with unique molecular identifiers (UMIs)



See what more we can do for **you** at **www.idtdna.com**.

IDT® INTEGRATED DNA TECHNOLOGIES

Introduction

This document outlines an example workflow for processing next generation sequencing data containing unique molecular identifiers, starting from FASTQ or similar raw data through making variant calls. The high level workflow is:

Construct an unmapped BAM tagged with UMIs
↓
Align reads
↓
Map BAM to consensus reads
↓
Produce variant calls from consensus reads

Note: This example workflow assumes that the sequencing run is demultiplexed using the standard methods recommended for your specific sequencing platform.

Referenced software packages

The following software packages are used in the examples within this document:

Package	Version	License	URL
Picard	2.9.0	MIT	https://github.com/broadinstitute/picard
bwa	0.7.15-r1140	GPL3	https://github.com/lh3/bwa
fgbio	0.2.1	MIT	https://github.com/fulcrumgenomics/fgbio
VarDict Java	—	MIT	https://github.com/AstraZeneca-NGS/VarDictJava

Note: Input of parameters for each tool within each package will affect analysis results. We recommend that you start running individual tools with the "-h" option to view a full list of parameters that you can tune.

A. Construct an unmapped BAM tagged with UMIs

The raw data format (e.g., FASTQ or BAM) and existing processing pipelines determine the optimal method to generate a mapped BAM file that contains UMI information. Ultimately, you should extract the in-line UMI sequence from each read and store it in the *RX* tag in an unmapped BAM file. Read 1 and read 2 of the same pair should have the same value stored in the *RX* tag, regardless of which read contains the UMI sequence.

fgbio's *ExtractUmisFromBam* processes unmapped BAM files, where the UMIs are contained within read 1 or read 2 sequences, and extracts the UMI sequences into the *RX* tag. Unmapped BAM files can be generated during the demultiplexing step. If raw data are in FASTQ files instead of unmapped BAM files, use Picard's *FastqToSam* tool to convert the FASTQ files into unmapped BAM files.

The read structure in the *ExtractUmisFromBam* command represents the duplex read structure. An example invocation follows:

```
java -Xmx4g -jar fgbio.jar ExtractUmisFromBam \
  --input=unmapped.bam --output=unmapped.withUMI.bam \
  --read-structure=3M2S146T 3M2S146T --molecular-barcode-tags=ZA ZB --single-tag=RX
```

In this invocation, the first "3M2S146T" represents the the structure of one strand, and the second "3M2S146T" represents the structure of the other strand:

- "3M" represents three UMI bases
- "2S" represents two skipped bases
- "146T" represents 146 bases in the read

B. Align reads

When you have an unmapped BAM file with *RX* tags, use a combination of an aligner and Picard's *MergeBamAlignment* tool to generate a mapped BAM that includes all necessary metadata. An example invocation follows:

```
java -Xmx4g -jar picard.jar SamToFastq I=unmapped.withUMI.bam F=/dev/stdout
INTERLEAVE=true \
    | bwa mem -p -t 8 hg38.fa /dev/stdin \
    | java -Xmx4g -jar picard.jar MergeBamAlignment \
       UNMAPPED=unmapped.withUMI.bam ALIGNED=/dev/stdin O=mapped.bam R=hg38.fa \
       SO=coordinate ALIGNER_PROPER_PAIR_FLAGS=true MAX_GAPS=-1 \
       ORIENTATIONS=FR VALIDATION_STRINGENCY=SILENT CREATE_INDEX=true
```

The next step produces consensus reads, making it unnecessary to duplicate-mark the reads or perform further processing of the raw data.

C. Map BAM to consensus reads

1. **Identify which reads come from the same source molecule** by using fgbio's *GroupReadsByUmi* tool, which assigns a unique source molecule ID to each applicable read, stores the ID in the *MI* tag, and outputs a BAM file that is sorted by the *MI* tag and ready for consensus calling. The source molecule is identified using a combination of UMI sequence and mapping positions from reads 1 and 2. An example invocation follows:

   ```
   java -Xmx4g -jar fgbio.jar GroupReadsByUmi \
     --input=mapped.bam --output=grouped.bam \
     --strategy=paired --edits=1 --min-map-q=20
   ```

   GroupReadsByUmi implements several strategies for matching UMIs to account for sequencing error. The adjacency method implements the directed adjacency graph method introduced by UMI-tools [1]. Parameters are available to control how many errors are allowed when matching UMIs at the same position and for filtering (i.e., ignoring) reads with low mapping quality. Low mapping quality reads should be ignored to prevent multiple consensus reads from being generated from multiple mismapped copies of the same source molecule.

2. **Combine each set of reads to generate consensus reads** using fgbio's *CallDuplexConsensusReads*. This step generates unmapped consensus reads from the output of *GroupReadsByUmi*. There are many parameters that affect the consensus calling; for an up-to-date listing and supporting documentation, run *CallDuplexConsensusReads* with the -h option. An example invocation follows with recommended parameters:

```
java -Djava.io.tmpdir=tmpDir -Xmx4g -jar fgbio.jar CallDuplexConsensusReads \
  --input=grouped.bam --output=consensus.unmapped.bam \
  --error-rate-pre-umi=45 --error-rate-post-umi=30 \
  --min-input-base-quality=30
```

   This script produces consensus reads for all molecules that have at least one observation.

3. **Filter consensus reads** using fgbio's *FilterConsensusReads*. There are two kinds of filtering: 1) masking or filtering individual bases in reads, and 2) filtering reads (i.e., not writing them to the output file). Base-level masking/filtering is only applied if per-base tags are present (see the documentation for *CallDuplexConsensusReads* for tag descriptions). Read-level filtering is always applied.

   When filtering reads, secondary alignments and supplementary records may be removed independently if they fail one or more filters. If either R1 or R2 primary alignments fail a filter, all records for the template will be filtered out. There are many parameters that affect the filtering of the consensus reads. For an up-to-date listing and supporting documentation, run *FilterConsensusRead* with the -h option. *FilterConsensusRead* can be applied to either mapped or unmapped BAM files. An example invocation with recommended parameters follows:

```
java -Djava.io.put=tmpDir -Xmx4g -jar fgbio.jar FilterConsensusReads \
  --input=consensus.unmapped.bam \
  --output=consensus.filtered.unmapped.bam \
  --ref=hg38.fa \
  --min-reads=6 3 3 \
  --max-read-error-rate=0.05 \
  --max-base-error-rate=0.1 \
  --min-base-quality=50 \
  --max-no-call-fraction=0.05
```

   This script produces a filtered, consensus BAM file containing sequences from molecules that have at least 3 reads for constructing the single-strand consensus and have at least 6 reads for constructing the duplex consensus.

D. Produce variant calls from consensus reads

After you have generated filtered, consensus reads, you must re-map the reads and call variants. The mapping procedure is the same as for raw reads:

```
java -Xmx4g -jar picard.jar SamToFastq I=consensus.unmapped.bam \
  F=/dev/stdout INTERLEAVE=true \
    | bwa mem -p -t 8 hg38.fa /dev/stdin \
    | java -Xmx4g -jar picard.jar MergeBamAlignment \
      UNMAPPED=consensus.unmapped.bam ALIGNED=/dev/stdin \
      O=consensus.mapped.bam R=hg38.fa \
      SO=coordinate ALIGNER_PROPER_PAIR_FLAGS=true MAX_GAPS=-1 \
      ORIENTATIONS=FR VALIDATION_STRINGENCY=SILENT CREATE_INDEX=true
```

Note: Duplicate marking should not be performed on consensus reads, because each read represents a unique source molecule.

To ensure that downstream processes, particularly variant calling, cannot double count evidence from the same template when both reads span a variant site in the same template, eliminate overlap between reads using fgbio's *ClipBam*. Clipping overlapping reads is only performed on "FR" read pairs and is implemented by clipping approximately half the overlapping bases from each read. By default, hard clipping is performed; soft-clipping may be substituted using the "--soft-clip" parameter.

```
java -Djava.io.tmpdir=/usr/tmp -Xmx4g -jar fgbio.jar ClipBam \
  --input=consensus.mapped.bam --output=consensus.mapped.clipped.bam \
  --ref=hg38.fa --soft-clip=false --overlapping-reads=true
```

Variant calling can be accomplished with the variant caller of your choice. The following example shows how to use *VarDictJava* in tumor-only mode to generate a VCF file, and Picard's *SortVcf* to sort and index the resulting VCF.

```
var_dict_dir=VarDictJava
min_af=0.01
tumor_name=tumor

$var_dict_dir/build/install/VarDict/bin/VarDict \
  -G hg38.fa \
  -N $tumor_name \
  -f $min_af
  -b consensus.mapped.clipped.bam \
  -z -c 1 -S 2 -E 3 -g 4 -th 4 \
  target_regions.bed \
    | $var_dict_dir/VarDict/teststrandbias.R \
    | $var_dict_dir/VarDict/var2vcf_valid.pl -N $tumor_name -E -f $min_af \
    | awk '{if ($1 ~ /^#/) print; else if ($4 != $5) print}' \
    > tmp.vcf
java -Xmx4g -jar picard.jar SortVcf I=tmp.vcf O=${tumor_name}.vcf
SD=hg38.dict && rm tmp.vcf
```

References

1. Smith T, Heger A, Sudbery I. (2017) UMI-tools: Modeling sequencing errors in Unique Molecular Identifiers to improve quantification accuracy. Genome Res, 27:491–499.

xGen® Duplex Seq Adapters—Tech Access: Processing sequence data with unique molecular identifiers (UMIs)

Revision history

Version	Date released	Description of changes
1.1	March 2018	Corrected typographical error (C.1, example invocation).
1	February 2018	Original analysis guideline