# Installation

## Introduction

1.1 The Mango/Big Data Genomics Ecosystem

## References

33

## Python Module Index

35

## Index

37
Mango is a distributed visualization tool that enables visualization of genomic data on top of Apache Spark.

1.1 The Mango/Big Data Genomics Ecosystem

Mango builds upon the open source Apache Spark, Apache Avro, and Apache Parquet projects. Additionally, Mango can be deployed for both interactive and production workflows using a variety of platforms.

1.1.1 Building Mango from Source

You will need to have Java 8, Apache Maven version 3.1.1 or later, and npm version 4.0.0 or later installed in order to build Mango.

Note: The default configuration is for Hadoop 2.7.3. If building against a different version of Hadoop, please pass \(-Dhadoop.version=<HADOOP_VERSION>\) to the Maven command. Mango will cross-build for both Spark 1.x and 2.x, but builds by default against Spark 1.6.3. To build for Spark 2, run the \(./scripts/move_to_spark2.sh\) script.

```
git clone https://github.com/bigdatagenomics/mango.git
cd mango
export MAVEN_OPTS="-Xmx512m -XX:MaxPermSize=128m"
mvn clean package -DskipTests
```

```
... BUILD SUCCESS
[INFO] --- maven-clean-plugin:3.5.0:clean (default-cli) @ mango ...
[INFO] Total time: 04:30 min
[INFO] Finished at: 2017-12-11T10:35:57-08:00
[INFO] Final Memory: 61M/1655M
```

Outputs
Running Mango

Mango is packaged as an überjar and includes all necessary dependencies, except for Apache Hadoop and Apache Spark.

Building for Python

To build and test Mango’s Python bindings, first set environmental variables pointing to the root of your Mango and Spark installation directories.

```
export SPARK_HOME=<PATH_TO_SPARK>
export MANGO_HOME=<PATH_TO_MANGO>
```

Next, build Mango jars without running tests, by running the following command from the root of the Mango repo install directory:

```
mvn clean package -DskipTests
```

Additionally, the PySpark dependencies must be on the Python module load path and the Mango JARs must be built and provided to PySpark. This can be done with the following bash commands:

```
PY4J_ZIP="$(ls -l "$SPARK_HOME/python/lib" | grep py4j)
export PYTHONPATH=$SPARK_HOME/python:$SPARK_HOME/python/lib/$PY4J_ZIP:
ASSEMBLY_DIR="/tmp/mango-assembly/target"
ASSEMBLY_JAR="$(ls -l "$ASSEMBLY_DIR/mango-assembly[0-9A-Za-z\_.-]*.jar" | grep -v javadoc | grep -v sources || true)
export PYSPARK_SUBMIT_ARGS="--jars "$ASSEMBLY_DIR/$ASSEMBLY_JAR --driver-class-path $ASSEMBLY_DIR/$ASSEMBLY_JAR pyspark-shell"
```

Next, install dependencies using the following commands:

```
cd mango-python
make prepare
cd ..
cd mango-viz
make prepare
cd ..
```

Finally, run `mvn package` again, this time enabling the python profile as well as tests:

```
mvn package -P python
```

This will enable the mango-python and mango-viz module as part of the Mango build. This module uses Maven to invoke a Makefile that builds a Python egg and runs tests.

1.1.2 Running Mango from Distribution

Fetching Mango Distribution

Mango is packaged as an überjar and includes all necessary dependencies, except for Apache Hadoop and Apache Spark.

To fetch the Mango distribution, run:
From the distribution directory, you can run Mango notebook or Mango browser:

First, make sure your SPARK_HOME env variable is set:

   export SPARK_HOME=<PATH_TO_SPARK>

Then run Mango notebook or Mango browser:

   cd mango-distribution-$(VERSION)
   ./bin/mango-notebook
   ./bin/mango-submit

### Installing python modules

To run Mango in a python notebook, install bdgenomics.mango.pileup, a Jupyter Widget:

   pip install bdgenomics.mango.pileup
   jupyter nbextension enable --py --sys-prefix bdgenomics.mango.pileup  # can be skipped for notebook version 5.3 and above

And bdgenomics.mango:

   pip install bdgenomics.mango

#### 1.1.3 Jupyter Widget Usage

**Note:** Python 2.7 is dropping support January 1, 2020. For this reason, Mango no longer supports Python 2.

The Mango widgets are Jupyter widgets built using pileup.js. The widgets support visualizations for alignments, features, variants, and genotypes in a Jupyter Notebook.

**Installation**

First, install and enable bdgenomics.mango.pileup, a Jupyter Widget:

   pip install bdgenomics.mango.pileup

   jupyter nbextension enable --py widgetsnbextension
   jupyter nbextension install --py --symlink --user bdgenomics.mango.pileup
   jupyter nbextension enable bdgenomics.mango.pileup --user --py

These tutorials show how to create a Jupyter pileup.js widget. An example notebook can be found in the Mango Github repository.
Pileup Example

This example shows how to visualize alignments through a Jupyter widget. It is located in `<PATH_TO_MANGO>/mango-viz/examples/pileup-tutorial.ipynb`.

```python
# imports
import bdgenomics.mango.pileup as pileup
from bdgenomics.mango.pileup.track import *
import pandas as pd

# read in JSON
readsJson = pd.read_json("./data/alignments.ga4gh.chr17.1-250.json")
GA4GHAlignmentJson = readsJson.to_json()

# make pileup track
tracks=[Track(viz="pileup", label="my Reads", source=pileup.sources.
  →GA4GHAlignmentJson(GA4GHAlignmentJson))]

# render tracks in widget
reads = pileup.PileupViewer(locus="chr17:1-100", reference="hg19", tracks=tracks)
reads
```

Variant Example

This example shows how to visualize variants through a Jupyter widget.

```python
# make variant track
tracks=[Track(viz="variants", label="my Variants", source=pileup.sources.
  →VcfDataSource("<path_to_file>/my_vcf.vcf"))]

# render tracks in widget
variants = pileup.PileupViewer(locus="chr1:10436-10564", reference="hg19", tracks=tracks)
variants
```
Feature Example

This example shows how to visualize features through a Jupyter widget.

```python
featuresJson = pd.read_json("./data/features.ga4gh.chr1.120000-125000.json")
GA4GHFeatureJson = featuresJson.to_json()

# make feature track
tracks=[Track(viz="features", label="my Features", source=pileup.sources.
       →GA4GHFeatureJson(GA4GHFeatureJson))]

# render tracks in widget
features = pileup.PileupViewer(locus='chr1:120000-121000', reference="hg19",
                               →tracks=tracks)
features
```

Genotype Example

This example shows how to visualize genotypes through a Jupyter widget.

```python
# make genotype track
tracks=[Track(viz="genotypes", label="my Genotypes", source=pileup.sources.
         →VcfDataSource("<path_to_file>/my_vcf.vcf"))]

# render tracks in widget
genotypes = pileup.PileupViewer(locus="chr22:21340030-21340150", reference="hg19",
                               →tracks=tracks)
genotypes
```
1.1.4 Mango Browser Examples

Mango browser is an HTML based genome browser that runs on local, remote, and cloud staged files. The Mango Browser utilizes Apache Spark and scalatra.

See file support for file types that are supported in the Mango Browser.

Mango Browser Options

The Mango browser uses the mango-submit script to start an Apache Spark session and launch the Mango Browser. The mango-submit script can be found in the Mango Distribution or in the Mango Github repository.

To see options that can be run with the Mango Submit script, run:

```
./mango-submit -h
```

You will see a list of options:

```
genome : Path to compressed .
--genome file. To build a new genome file, run bin/make_genome.
-cacheSize N : Bp to cache on
--driver.
--coverage VAL : A list of coverage
--files to view, separated by commas (,)
--debugFrontend : For debugging
--purposes. Sets front end in source code to avoid recompilation.
--discover : This turns on
--discovery mode on start up.
--features VAL : The feature files
--to view, separated by commas (,)
-h (-help, --help, -?) : Print help
--parquetIsBinned
--binned parquet pre-fetch warmup step
--parquet_block_size N : Parquet block size
--(default = 128mb)
--parquet_compression_codec [UNCOMPRESSED | SNAPPY | GZIP | LZO] : Parquet compression
(continues on next page)
```
-parquet_disable_dictionary encoding : Disable dictionary encoding
-parquet_logging_level VAL level (default = severe) : Parquet logging level (default = severe)
-parquet_page_size N (default = 1mb) : Parquet page size
-port N : The port to bind to
--for visualization. The default is 8080. : The port to bind to
--prefetchSize N : Bp to prefetch in executors.
--preload VAL : Chromosomes to prefetch, separated by commas (,).
--print_metrics the log on completion : Print metrics to the log on completion
--reads VAL : A list of reads files to view, separated by commas (,)
--repartition : Repartitions data to default number of partitions.
--test : For debugging
--variants VAL : A list of variants files to view, separated by commas (,). Vcf files require a corresponding tbi
--index.

Note that a genome file is always required when running the Mango Browser. See how to build a genome.

Running Mango Browser Examples Locally

The Mango Github repository contains example scripts and data files for running Mango browser on region chr17:7500000-7515000 of a single alignement sample. Once Mango is built, you can run the following command from the root mango directory to view Mango browser:

```bash
./example-files/browser-scripts/run-example.sh
```

This script contains the following command:

```bash
bin/mango-submit ./example-files/hg19.genome
--reads ./example-files/chr17.7500000-7515000.sam
--variants ./example-files/ALL.chr17.7500000-7515000.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf
```

This script specifies the required genome reference file:

```bash
./example-files/hg19.genome
```

An optional alignment file:

```bash
--reads ./example-files/chr17.7500000-7515000.sam
```

An optional variant file:

```bash
--variants ./example-files/ALL.chr17.7500000-7515000.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf
```
Once the example script is running, navigate to localhost:8080 to view the Mango browser. Navigate to chr17:7500000-7515000 to view data.

There is also another example script at ./example-files/browser-scripts/run-http-example.sh that runs on remote files from Amazon S3.

This script contains the following command:

```
bin/mango-submit ./example-files/hg19.genome \
  -variants http://s3.amazonaws.com/1000genomes/phase1/analysis_results/integrated_\n  →call_sets/ALL.chr1.integrated_phase1_v3.20101123.snps_indels_svs.genotypes.vcf.gz \
  -reads http://s3.amazonaws.com/1000genomes/phase1/data/NA19661/exome_alignment/\n  →NA19661.mapped.illumina.mosaik.MXL.exome.20110411.bam
```

### Running Mango Browser with Parameters

Mango can accept Apache Spark parameters, as well as Mango parameters shown above.

To run Mango browser with user specified Apache Spark parameters, run

```
./bin/mango-submit <Spark-parameters> -- <Mango-parameters>
```

<Spark-parameters> include Apache Spark specific configuration settings. 
<Mango-parameters> are shown in the output of ./bin/mango-submit.

Note that a genome file is required to run the Mango browser.

### Running example files on a cluster with HDFS

The Mango browser can run files that are staged on Hadoop Distributed File System (HDFS).

To run the example files on a cluster with hdfs, first put example-files on hdfs:

```
hdfs dfs -put example-files
```

Then, run mango-submit:

```
./bin/mango-submit ./example-files/hg19.genome \
  -genes http://www.biodalliance.org/datasets/ensGene.bb \
  -reads hdfs://<path_to_examples>/example-files/chr17.7500000-7515000.sam \
  -variants hdfs://<path_to_examples>/example-files/ALL.chr17.7500000-7515000.\n  →phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf
```

### Running on Apache YARN

YARN is a resource management system for clusters. The Mango browser can run on YARN clusters, and requires package org.apache.parquet:parquet-avro:1.8.3. To run the Mango browser on YARN, include parquet-avro as a package on start-up:

```
./bin/mango-submit --packages org.apache.parquet:parquet-avro:1.8.3 \n  --master yarn-client \n  <mango-parameters>
```
1.1.5 Building a Genome

The Mango browser requires a remote TwoBit reference file, a local gene file, and a local chromosome size file to run. These files are downloaded from UCSC Genome Browser downloads.

Assembling a Default Genome File

To assemble a genome file, run:

```
./bin/make_genome <GENOME_NAME> <OUTPUT_LOCATION>
```

This will download required files for the genome build, compress them, and save them to `<OUTPUT_LOCATION>`. All builds accessible from the UCSC Downloads page are supported.

The file `hg19.genome` build exists in the `example-files/` folder as a reference.

Assembling a Custom Genome File

If you need to assemble a custom genome file that is not supported by the `make_genome` executable, you can assemble one as follows.

First specify a folder, `<YOUR_GENOME>.genome`, and include the following files:

- `cytoBand.txt`: UCSC formatted cytoBand file
- `<YOUR_GENOME>.chrom.sizes`: tab delimited file of chromosome names and sizes
- `refGene.txt`: UCSC formatted tab delimited text file of gene information
- `properties.txt`: File containing metainformation for your genome

The `properties.txt` file should be formatted as follows:

```
sequenceLocation=http://<path_to_remote_2bit_file>.2bit
refGene=refGene.txt
chromSizes=<YOUR_GENOME>.chrom.sizes
cytoband=cytoBand.txt
```

Note that the `sequenceLocation` parameter must link to a remote TwoBit file.

1.1.6 Mango Python Examples

Note: Python 2.7 is dropping support January 1, 2020. For this reason, Mango no longer supports Python 2.

Running Mango Notebook Locally

Once Mango and Mango python is built, you can run the following command to view Mango notebook:

```
./bin/mango-notebook
```

Mango notebook depends on Jupyter notebook. To install all dependencies for Mango notebook in a virtual environment, see installation instructions.

One Mango notebook is running, you can view local results at localhost:<port>, where `<port>` is the open port assigned by Jupyter notebook. There are three notebooks that can be viewed as examples in the Mango repository:
Running the Mango Notebook with Parameters

The Mango Notebook can be run with Apache Spark parameters and Jupyter notebook parameters. To run Mango notebook with user specified parameters, run

```shell
./bin/mango-notebook <Spark-parameters> -- <Jupyter-notebook-parameters>
```

Running the Mango Notebook on YARN

YARN is a resource management system for clusters. The Mango notebook can run on YARN clusters, and requires jars for org.apache.parquet:parquet-hadoop:1.8.3. To run the Mango browser on YARN, download parquet-hadoop jar:

Then include the jar in spark.driver.extraClassPath:

```shell
wget http://central.maven.org/maven2/org/apache/parquet/parquet-hadoop/1.8.3/parquet-hadoop-1.8.3.jar
```

```shell
./bin/mango-notebook --master yarn \
--conf spark.driver.extraClassPath=<path_to_jar>/parquet-hadoop-1.8.3.jar \
-- <Jupyter-notebook-parameters>
```

1.1.7 Mango Python API Documentation

bdgenomics.mango Package

bdgenomics.mango provides hooks for visualizing genomic data on top of Apache Spark. Available visualizations are supported for alignments, coverage, variants and features.

Alignments

```
<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlignmentSummary</td>
<td>AlignmentSummary class.</td>
</tr>
<tr>
<td>FragmentDistribution</td>
<td>FragmentDistribution class.</td>
</tr>
<tr>
<td>MapQDistribution</td>
<td>MapQDistribution class.</td>
</tr>
<tr>
<td>IndelDistribution</td>
<td>IndelDistribution class.</td>
</tr>
</tbody>
</table>
```

bdgenomics.mango.alignments.AlignmentSummary

```python
class bdgenomics.mango.alignments.AlignmentSummary:
```

AlignmentSummary class. AlignmentSummary provides scrollable visualization of alignments based on ge-
nomic regions.

```python
__init__(spark, ac, alignmentDataset, sample=1.0)
```

Initializes an AlignmentSummary class.

**Args:**

- **param spark** SparkSession
- **param ac** ADAMContext
- **param alignmentDataset** bdgenomics.adam.Dataset.AlignmentRecoDataset
- **param sample** fraction of reads to sample from

**Methods**

- `__init__(spark, ac, alignmentDataset[, sample])` Initializes an AlignmentSummary class.
- `getCoverageDistribution([bin_size])` Computes coverage distribution for this AlignmentDataset.
- `getFragmentDistribution()` Computes fragment distribution for this AlignmentDataset.
- `getIndelDistribution([bin_size])` Computes insertion and deletion distribution for this AlignmentDataset.
- `getMapQDistribution()` Computes mapping quality distribution for this AlignmentDataset.
- `viewPileup(contig, start, end[, reference, ...])` Visualizes a portion of this AlignmentDataset in a scrollable pileup widget

### bdgenomics.mango.alignments.FragmentDistribution

**class** `bdgenomics.mango.alignments.FragmentDistribution(ss, alignmentDataset, sample=1.0)`

FragmentDistribution class. Plotting functionality for visualizing fragment distributions of multi-sample cohorts.

```python
__init__(ss, alignmentDataset, sample=1.0)
```

Initializes a FragmentDistribution class. Computes the fragment distribution of a AlignmentDataset. This Dataset can have data for multiple samples.

**Args:**

- **param ss** global SparkSession.
- **param alignmentDataset** bdgenomics.adam.Dataset.AlignmentDataset
- **param sample** Fraction to sample AlignmentDataset. Should be between 0 and 1

**Methods**

- `__init__(ss, alignmentDataset[, sample])` Initializes a FragmentDistribution class.
- `plotDistributions([normalize, cumulative, ...])` Plots final distribution values and returns the plotted distribution as a Counter object.
Attributes

- pre_sampled
- rdd
- sample
- seed
- ss

**bdgenomics.mango.alignments.MapQDistribution**

class bdgenomics.mango.alignments.MapQDistribution(ss, alignmentDataset, sample=1.0)

MapQDistribution class. Plotting functionality for visualizing mapping quality distributions of multi-sample cohorts.

__init__(ss, alignmentDataset, sample=1.0)

Initializes a MapQDistribution class. Computes the mapping quality distribution of an AlignmentDataset. This Dataset can have data for multiple samples.

Args:

- param ss global SparkSession.
- param alignmentDataset A bdgenomics.adam.dataset.AlignmentDataset object.
- param sample Fraction to sample AlignmentDataset. Should be between 0 and 1

Methods

__init__(ss, alignmentDataset[, sample])

Initializes a MapQDistribution class.

plotDistributions([normalize, cumulative, ...])

Plots final distribution values and returns the plotted distribution as a Counter object.

Attributes

- pre_sampled
- rdd
- sample
- seed
- ss

**bdgenomics.mango.alignments.IndelDistribution**

class bdgenomics.mango.alignments.IndelDistribution(ss, alignmentDataset, sample=1.0, bin_size=1000000)

IndelDistribution class. IndelDistribution calculates indel distributions on an AlignmentDataset.

__init__(ss, alignmentDataset, sample=1.0, bin_size=1000000)

Initializes a IndelDistribution class. Computes the insertiona and deletion distribution of alignmentDataset.

Args:
### IndelDistribution

**param SparkSession** the global SparkSession  
**param alignmentDataset** A bdgenomics.adam.dataset.AlignmentDataset object  
**param bin_size** Division size per bin

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>(ss, alignmentDataset[, sample, ...])</td>
<td>Initializes a IndelDistribution class.</td>
</tr>
<tr>
<td>plot([testMode, plotType])</td>
<td>Plots final distribution values and returns the plotted distribution as a counter object.</td>
</tr>
</tbody>
</table>

### Coverage

**CoverageDistribution(ss, coverageDataset[, sample, ...])** CoverageDistribution class.

**bdgenomics.mango.coverage.CoverageDistribution**

**class** bdgenomics.mango.coverage.CoverageDistribution(ss, coverageDataset[, sample=1.0, bin_size=10, pre_sampled=False])  
CoverageDistribution class. Plotting functionality for visualizing coverage distributions of multi-sample cohorts.  
**__init__(ss, coverageDataset, sample=1.0, bin_size=10, pre_sampled=False)**  
Initializes a CoverageDistribution class. Computes the coverage distribution of a CoverageRDD. This RDD can have data for multiple samples.  
**Args:**  
**param ss** global SparkSession.  
**param coverageRDD** bdgenomics.adam.rdd.CoverageDataset  
**param sample** Fraction to sample CoverageRDD. Should be between 0 and 1  
**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>(ss, coverageDataset[, sample, ...])</td>
<td>Initializes a CoverageDistribution class.</td>
</tr>
<tr>
<td>plotDistributions([normalize, cumulative, ...])</td>
<td>Plots final distribution values and returns the plotted distribution as a Counter object.</td>
</tr>
</tbody>
</table>

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre_sampled</td>
<td></td>
</tr>
<tr>
<td>rdd</td>
<td></td>
</tr>
<tr>
<td>sample</td>
<td></td>
</tr>
<tr>
<td>seed</td>
<td></td>
</tr>
<tr>
<td>ss</td>
<td></td>
</tr>
</tbody>
</table>
Features

`bdgenomics.mango.feature.FeatureSummary(ac, dataset)` FeatureSummary class.

`bdgenomics.mango.features.FeatureSummary` class `bdgenomics.mango.features.FeatureSummary(ac, dataset)` FeatureSummary class. FeatureSummary provides scrollable visualization of features based on genomic regions.

`__init__(ac, dataset)` Initializes a GenomicRDD viz class.

**Args:**

- `param ac` bdgenomics.adam.damContext.ADAMContext
- `param dataset` bdgenomics.adam.rdd.FeatureDataset

**Methods**

`__init__(ac, dataset)` Initializes a GenomicRDD viz class.

`viewPileup(contig, start, end[, reference, ...])` Visualizes a portion of this FeatureDataset in a scrollable pileup widget

Variants

`bdgenomics.mango.variants.VariantSummary(ac, dataset)` VariantSummary class.

`bdgenomics.mango.variants.VariantSummary` class `bdgenomics.mango.variants.VariantSummary(ac, dataset)` VariantSummary class. VariantSummary provides scrollable visualization of variants based on genomic regions.

`__init__(ac, dataset)` Initializes a GenomicDataset viz class.

**Args:**

- `param ac` bdgenomics.adamContext.ADAMContext
- `param dataset` bdgenomics.adam.rdd.VariantDataset

**Methods**

`__init__(ac, dataset)` Initializes a GenomicDataset viz class.

`viewPileup(contig, start, end[, reference, ...])` Visualizes a portion of this VariantRDD in a scrollable pileup widget
1.1.8 Jupyter Widget API Documentation

bdgenomics.mango.pileup Package

Each widget instance calls the PileupViewer to draw an interactive widget for genomic data.

PileupViewer

PileupViewer

Create a new Mock object.

bdgenomics.mango.pileup.pileupViewer.PileupViewer

bdgenomics.mango.pileup.pileupViewer.PileupViewer = <Mock name='mock.register()' id='139969371636560'>

Create a new Mock object. Mock takes several optional arguments that specify the behaviour of the Mock object:

• spec: This can be either a list of strings or an existing object (a class or instance) that acts as the specification for the mock object. If you pass in an object then a list of strings is formed by calling dir on the object (excluding unsupported magic attributes and methods). Accessing any attribute not in this list will raise an AttributeError.

  If spec is an object (rather than a list of strings) then mock.__class__ returns the class of the spec object. This allows mocks to pass isinstance tests.

• spec_set: A stricter variant of spec. If used, attempting to set or get an attribute on the mock that isn’t on the object passed as spec_set will raise an AttributeError.

• side_effect: A function to be called whenever the Mock is called. See the side_effect attribute. Useful for raising exceptions or dynamically changing return values. The function is called with the same arguments as the mock, and unless it returns DEFAULT, the return value of this function is used as the return value.

  Alternatively side_effect can be an exception class or instance. In this case the exception will be raised when the mock is called.

  If side_effect is an iterable then each call to the mock will return the next value from the iterable. If any of the members of the iterable are exceptions they will be raised instead of returned.

• return_value: The value returned when the mock is called. By default this is a new Mock (created on first access). See the return_value attribute.

• wraps: Item for the mock object to wrap. If wraps is not None then calling the Mock will pass the call through to the wrapped object (returning the real result). Attribute access on the mock will return a Mock object that wraps the corresponding attribute of the wrapped object (so attempting to access an attribute that doesn’t exist will raise an AttributeError).

  If the mock has an explicit return_value set then calls are not passed to the wrapped object and the return_value is returned instead.

• name: If the mock has a name then it will be used in the repr of the mock. This can be useful for debugging. The name is propagated to child mocks.

Mocks can also be called with arbitrary keyword arguments. These will be used to set attributes on the mock after it is created.
Sources

Sources specify where the genomic data comes from. Sources can come from a url, a GA4GHDatasource, or a JSON string of GA4GH formatted data.

- **BamDataSource**(url[, indexUrl]) Initializes file source from bam file endpoint.
- **VcfDataSource**(url[, indexUrl]) Initializes file source from vcf file endpoint.
- **TwoBitDataSource**(url[, indexUrl]) Initializes file source from twoBit file endpoint.
- **BigBedDataSource**(url[, indexUrl]) Initializes file source from big bed (.bb) file endpoint.
- **GA4GHAlignmentJson**(json) Initializes GA4GH Alignment JSON.
- **GA4GHVariantJson**(json) Initializes GA4GH variant JSON.
- **GA4GHFeatureJson**(json) Initializes GA4GH feature JSON.
- **GA4GHAlignmentSource**(endpoint, readGroupId) Initializes GA4GHAlignmentSource.
- **GA4GHVariantSource**(endpoint, readGroupId[,...]) Initializes GA4GHVariantSource.
- **GA4GHFeatureSource**(endpoint, readGroupId[,...]) Initializes GA4GHFeatureSource.

**bdgenomics.mango.pileup.sources.BamDataSource**

```python
class bdgenomics.mango.pileup.sources.BamDataSource (url, indexUrl=None)
    Initializes file source from bam file endpoint.
```

**Args:**

- **param str** url to file
- **param str** indexUrl to index file

```python
__init__(url, indexUrl=None)
    Initializes file sources.
```

**Args:**

- **param str** url to file
- **param str** indexUrl to index file

**Methods**

```python
__init__(url[, indexUrl])
    Initializes file sources.
```

**Attributes**

```python
dict_
    name
    name that pileup.js uses to identify sources
```

**bdgenomics.mango.pileup.sources.VcfDataSource**

```python
class bdgenomics.mango.pileup.sources.VcfDataSource (url, indexUrl=None)
    Initializes file source from vcf file endpoint.
```
Args:

param str url to file
param str indexUrl to index file
__init__(url, indexUrl=None)
Initializes file sources.

Methods

__init__(url, indexUrl)
Initializes file sources.

Attributes

dict_
name

bdgenomics.mango.pileup.sources.TwoBitDataSource

class bdgenomics.mango.pileup.sources.TwoBitDataSource (url, indexUrl=None)
Initializes file source from twoBit file endpoint.

Args:

param str url to file
__init__(url, indexUrl=None)
Initializes file sources.

Args:

param str url to file
param str indexUrl to index file

Methods

__init__(url, indexUrl)
Initializes file sources.

Attributes

dict_
name
bdgenomics.mango.pileup.sources.BigBedDataSource

class bdgenomics.mango.pileup.sources.BigBedDataSource(url, indexUrl=None)
    Initializes file source from big bed (.bb) file endpoint.

    Args:
        url (str): url to file

    __init__ (url, indexUrl=None)
    Initializes file sources.

    Args:
        url (str): url to file
        indexUrl (str): indexUrl to index file

Methods

    __init__ (url, indexUrl)
    Initializes file sources.

Attributes

    dict_
    name
    name that pileup.js uses to identify sources

bdgenomics.mango.pileup.sources.GA4GHAlignmentJson

class bdgenomics.mango.pileup.sources.GA4GHAlignmentJson(json)
    Initializes GA4GH Alignment JSON.

    Args:
        json (str): json in GA4GH format

    __init__ (json)
    Initializes GA4GH JSON.

    Args:
        json (str): json in GA4GH format

Methods

    __init__ (json)
    Initializes GA4GH JSON.

Attributes

    dict_
    name
    name that pileup.js uses to identify sources
bdgenomics.mango.pileup.sources.GA4GHVariantJson

class bdgenomics.mango.pileup.sources.GA4GHVariantJson(json)
    Initializes GA4GH variant JSON.
    Args:
        param str json in GA4GH format

    __init__(json)
    Initializes GA4GH JSON.
    Args:
        param str json in GA4GH format

Methods

    __init__(json) Initializes GA4GH JSON.

Attributes

    dict_ name that pileup.js uses to identify sources

bdgenomics.mango.pileup.sources.GA4GHFeatureJson

class bdgenomics.mango.pileup.sources.GA4GHFeatureJson(json)
    Initializes GA4GH feature JSON.
    Args:
        param str json in GA4GH format

    __init__(json)
    Initializes GA4GH JSON.
    Args:
        param str json in GA4GH format

Methods

    __init__(json) Initializes GA4GH JSON.

Attributes

    dict_ name that pileup.js uses to identify sources
bdgenomics.mango Documentation, Release 0.0.3

bdgenomics.mango.pileup.sources.GA4GHAlignmentSource

```python
class bdgenomics.mango.pileup.sources.GA4GHAlignmentSource(endpoint, readGroupId, callSetIds=None):
    Initializes GA4GHAlignmentSource.
    Args:
    param str url endpoint
    param str read group id
    __init__(endpoint, readGroupId, callSetIds=None)
    Initializes GA4GHSource.
    Args:
    param str url endpoint
    param str read group id
    param str optional call set ID for variants

Methods

__init__(endpoint, readGroupId[, callSetIds]) Initializes GA4GHSource.

Attributes

dict_

name name that pileup.js uses to identify sources
```

bdgenomics.mango.pileup.sources.GA4GHVariantSource

```python
class bdgenomics.mango.pileup.sources.GA4GHVariantSource(endpoint, readGroupId, callSetIds=None):
    Initializes GA4GHSource.
    Args:
    param str url endpoint
    param str call set ID
    param str optional call set ID for variants
    __init__(endpoint, readGroupId, callSetIds=None)
    Initializes GA4GHSource.
    Args:
    param str url endpoint
    param str read group id
    param str optional call set ID for variants
```

Chapter 1. Introduction
Methods

```python
__init__(endpoint, readGroupId[, callSetIds])
```
Initializes GA4GHSource.

Attributes

```python
dict_
name
```
name that pileup.js uses to identify sources

**bdgenomics.mango.pileup.sources.GA4GHFeatureSource**

```python
class bdgenomics.mango.pileup.sources.GA4GHFeatureSource:
```
Initializes GA4GHFeatureSource.

**Args:**

```python
param str url endpoint
```

```python
__init__(endpoint, readGroupId, callSetIds=None)
```
Initializes GA4GHSource.

**Args:**

```python
param str url endpoint
param str read group id
param str optional call set ID for variants
```

Methods

```python
__init__(endpoint, readGroupId[, callSetIds])
```
Initializes GA4GHSource.

Attributes

```python
dict_
name
```
name that pileup.js uses to identify sources

**Track**

Tracks specify what visualization will be drawn.

```python
Track(**kwargs)
```
A trait for a pileupTrack, requires a viz string of (coverage, pileup, features, variants, genome, genes, scale, or location) and a DataSource.

```python
track_to_json(pyTrack, manager)
```
Serialize a Track.

```python
track_from_json(js, manager)
```
Deserialize a Track from JSON.

```python
tracks_to_json(pyTracks, manager)
```
Serialize a Python date object.

Continued on next page
Table 37 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tracks_from_json</code></td>
<td>Deserialize a list of Tracks from JSON.</td>
</tr>
</tbody>
</table>

### bdgenomics.mango.pileup.track.Track

#### Class

**Class** `bdgenomics.mango.pileup.track.Track(**kwargs)`

A trait for a pileupTrack, requires a viz string of (coverage, pileup, features, variants, genome, genes, scale, or location) and a DataSource.

**__init__(**kwargs)***

Initializes track.

**Args:**

- **param kwarg** Should contain viz, optional source, optional label.

#### Methods

- **__init__(**kwargs)**: Initializes track.
- **class_init(cls, name)**: Part of the initialization which may depend on the underlying HasDescriptors class.
- **default_value_repr()**: DEPRECATED: Retrieve the static default value for this trait.
- **error(obj, value)**
- **get(obj[, cls])**
- **get_default_value()**: DEPRECATED: Retrieve the static default value for this trait.
- **get_metadata(key[, default])**: DEPRECATED: Get a metadata value.
- **info()**: DEPRECATED: Set the static default value for the trait type.
- **instance_init(obj)**: Part of the initialization which may depend on the underlying HasDescriptors instance.
- **set(obj, value)**
- **set_metadata(key, value)**: DEPRECATED: Set a metadata key/value.
- **tag(**metadata)**: Sets metadata and returns self.

#### Attributes

- **allow_none**
- **default_value**
- **info_text**
- **label**: Label for this track.
- **metadata**
- **name**
- **read_only**
- **source**: Datasource consumed by pileup.js.
- **sourceOptions**: Options for source.
- **this_class**
- **viz**: visualization specified in pileup.js.
bdgenomics.mango.pileup.track.track_to_json

Serialize a Track. Attributes of this dictionary are to be passed to the JavaScript Date constructor.

Args:

param Track  Track object

param any  manager. Used for widget serialization.

Returns:  dict of Track elements (viz, source, sourceOptions and label)

bdgenomics.mango.pileup.track.track_from_json

Deserialize a Track from JSON.

Args:

param (str)  json for Track containing viz, source, sourceOptions and label

param (any)  manager. Used for widget serialization.

Returns:  Track: pileup Track built from json

bdgenomics.mango.pileup.track.tracks_to_json

Serialize a Python date object. Attributes of this dictionary are to be passed to the JavaScript Date constructor.

Args:

param (List)  List of Tracks

param (any)  manager. Used for widget serialization.

Returns:  List of dict of Track elements (viz, source, sourceOptions and label)

bdgenomics.mango.pileup.track.tracks_from_json

Deserialize a list of Tracks from JSON.

Args:

param (str)  json for list of Tracks containing viz, source, sourceOptions and label

param (any)  manager. Used for widget serialization.

Returns:  List: List of pileup Track built from json

1.1.9 Supported Files

Supported File Types

Mango supports the following file types:
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Supported File Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alignments</td>
<td>Parquet, bam, indexed bam, sam</td>
</tr>
<tr>
<td>Variants</td>
<td>Parquet, vcf, indexed vcf, vcf.gz</td>
</tr>
<tr>
<td>Features</td>
<td>Parquet, bed, narrowPeak</td>
</tr>
<tr>
<td>Genome</td>
<td>Parquet, twoBit*, fa, fasta</td>
</tr>
</tbody>
</table>

*TwoBit files must be staged locally for access.

**Accessing http files through Mango**

Mango can copy and read http files. To do so, when running `mango-submit`, set `spark.local.dir` to a path in the user’s home directory:

```
./bin/mango-submit
   --conf spark.local.dir=<user home>/spark-tmp
```

This will allow Spark to access temporary http files.

**Accessing s3a files through Mango**

To access s3a files when running on AWS, you need the `net.fnothaft:jsr203-s3a` package, and the bam splitter to be enabled:

```
./bin/mango-submit 
   --packages org.apache.parquet:parquet-avro:1.8.2 
   --packages net.fnothaft:jsr203-s3a:0.0.2 
   --conf spark.hadoop.hadoopbam.bam.enable-bai-splitter=true 
   -- hg19.2bit 
   -reads s3a://1000genomes/phase1/data/NA12878/exome_alignment/NA12878.mapped.illumina.mosaik.CEU.exome.20110411.bam
```

### 1.1.10 Running Mango from Docker

Running Mango from Docker requires Docker to be installed.

Mango is available on Docker through UCSC cgl and has open code available at cgl-docker-lib.

**Running Mango Browser on Docker**

To run Mango browser example files on Linux in Docker run:

```
docker run -it
   --net=host 
   -p 8080:8080 
   quay.io/ucsc_cgl/mango:latest 
   -- /opt/cgl-docker-lib/mango/example-files/hg19.genome 
   -reads /opt/cgl-docker-lib/mango/example-files/chr17.7500000-7515000.sam.adam 
   -variants /opt/cgl-docker-lib/mango/example-files/ALL.chr17.7500000-7515000.
   --phase3_shapeit2_mvnncall_integrated_v5a.20130502.genotypes.vcf
```
Note: To run the Mango browser on OS X, remove --net=host.

To run Mango browser on local data, you must first mount these files with the Docker -v flag. For example, if you have local files stored at `<example-file-path>:

```bash
docker run -it -p 8080:8080 \
   -v <example-file-path>:<docker-container-path> \
   quay.io/ucsc_cgl/mango:latest \
   -- <docker-container-path>/hg19.genome \
   -reads <docker-container-path>/chr17.7500000-7515000.sam.adam \
   -variants <docker-container-path>/ALL.chr17.7500000-7515000.phase3_shapeit2_\n   --mvnca1l_integrated_v5a.20130502.genotypes.vcf
```

To create a reference, see Building a Genome.

### Running Mango Notebook on Docker

To run Mango notebook on Linux in Docker run:

```bash
docker run --net=host -it -p 8888:8888 \
   --entrypoint=/opt/cgl-docker-lib/mango/bin/mango-notebook \
   quay.io/ucsc_cgl/mango:latest \
   -- --ip=0.0.0.0 --allow-root
```

Note: To run the Mango notebook on OS X, remove --net=host.

To view a number of ipython notebook examples, navigate to: /opt/cgl-docker-lib/mango/example-files.

### 1.1.11 Running Mango on Google Cloud

Cloud Dataproc provides pre-built Hadoop and Spark distributions which allows users to easily deploy and run Mango.

This documentation explains how to configure requirements to connect with Google Cloud on your local machine, and how to run Mango on GCP.

#### Before you Start

Commands in this section will require users to:

1. Create an account on Google Cloud
2. Install the gcloud cli

#### Creating a Dataproc Cluster

The necessary initialization scripts are available at the Google Cloud Storage bucket located at gs://mango-initialization-bucket/

In order to access this bucket through the cloud dataproc cluster, it is necessary to give billing account manager permissions to dataproc-accounts.iam.gserviceaccount.com and compute@developer.gserviceaccount.com through the IAM console.

Create the Cloud Dataproc Cluster (modify the fields as appropriate) with the below installation script:
gcloud dataproc clusters create <cluster-name> \
   --project <project_id> \
   --bucket <optional_bucket_name> \
   --metadata MINICONDA_VARIANT=2 \
   --master-machine-type=n1-standard-1 \
   --worker-machine-type=n1-standard-1 \
   --master-boot-disk-size=50GB \
   --worker-boot-disk-size=10GB \
   --initialization-actions \
      gs://mango-initialization-bucket/install.sh

After the above steps are completed, ssh into the master node with the following command:

gcloud compute ssh <cluster-name>-m

1.1.12 Running Mango through Docker

Before Mango can run, it is recommended to stage datasets into HDFS if you are trying to view specific files. The created container will share the same hadoop file system with the root master user.

hdfs dfs -put /<local-machine-path> /<hdfs-path>

An example docker startup script is available in the Mango gce scripts directory for running mango notebook, or for running mango browser [root permissions may be necessary for docker].

wget 'https://raw.githubusercontent.com/bigdatagenomics/mango/master/bin/gce/run-notebook.sh'
bash run-notebook.sh --entrypoint=/opt/cgl-docker-lib/mango/bin/mango-notebook

Once the notebook is running, connect to Mango by setting up a tunnel to your local computer via the exposed port in the master node:

gcloud compute ssh <cluster-name>-m -- -N -L localhost:<local-port>:localhost:8888

You can navigate to the notebook through your local browser by pointing it towards http://localhost:<local-port>/.

Once in the browser notebook environment, navigate to /opt/cgl-docker-lib/mango/example-files/ to try out the example files after configuring the file paths to read relative to the home directory in HDFS.

Public datasets can be accessed by referencing google cloud storage at gs://genomics-public-data/.

More information about available public datasets on Google cloud can be found online

More information on using the dataproc cluster’s Spark interface is available through Google Cloud documentation

1.1.13 Running Mango from Amazon EMR

Amazon Elastic Map Reduce (EMR) provides pre-built Hadoop and Spark distributions which allows users to easily deploy and run Mango. This documentation explains how to configure requirements to connect with AWS on your local machine, and how to run Mango on AWS.
Before you Start

You will first need to set up an EC2 key pair and configure the AWS Command Line Interface (AWS CLI) using your key. This will allow you to create clusters and ssh into your machine from the command line.

First, follow the following configuration steps:

1. Set up an EC2 key pair. This keypair allows you to securely access instances on AWS using a private key.
2. Install the AWS CLI
3. Configure the AWS CLI

This will allow you to access AWS using your credentials.

1.1.14 Running Mango through Docker

This section explains how to run the Mango browser and the Mango notebook through Docker on Amazon EMR. Using Docker allows users to quickly get Mango up and running without needing to configure different environment variables on their cluster. If you need more flexibility in configuration, see Running Mango standalone.

Creating a Cluster

First, you must configure an EMR cluster. This can be done using the AWS CLI.

Through the command line, create a new cluster:

```
aws emr create-cluster
--release-label emr-5.24.0
--name 'emr-5.24.0 Mango example'
--applications Name=Hadoop Name=Hive Name=Spark
--ec2-attributes KeyName=<YOUR_EC2_KEY>,InstanceProfile=EMR_EC2_DefaultRole
--service-role EMR_DefaultRole
--instance-groups
    InstanceGroupType=MASTER,InstanceCount=1,InstanceType=c3.4xlarge
    InstanceGroupType=CORE,InstanceCount=4,InstanceType=c3.4xlarge
--region <your_region>
--log-uri s3://<your-s3-bucket>/emr-logs/
--bootstrap-actions
    Name='Install Mango', Path="s3://bdg-mango/install-bdg-mango-docker-emr5.sh"
```

In the code chunk above, set your EC2 key pair name:

```
--ec2-attributes KeyName=<YOUR_EC2_KEY>
```

Note the instance counts:

```
InstanceGroupType=CORE,InstanceCount=4,InstanceType=c3.4xlarge
```

In this example, we have set the number of instance counts, or the number of workers, to 4. If you are using larger or smaller workloads, you should scale this number up or down accordingly. Note that more instances will cost more money.

The bootstrap action:

```
--bootstrap-actions
    Name='Install Mango', Path="s3://bdg-mango/install-bdg-mango-docker-emr5.sh"
```
will download docker and required scripts. These scripts will be available on your EMR master node in the directory /home/hadoop/mango-scripts.

Enabling a Web Connection

To view the Spark UI, notebook, and browser, you must setup a web connection for the cluster. To do so, navigate to your Amazon EMR Clusters page, click your started cluster, and click on Enable Web Connection and follow the instructions for enabling a connection.

These instructions will require you to install the FoxyProxy addon in your web browser. Note that for accessing the recommended 8157 port using FoxyProxy (as well as port 22 for ssh), you will have to expose these ports in the security group for the master node.

To expose required ports on the master node, navigate to Security and access in your Cluster EMR manager. Click on Security groups for Master. Add a inbound new rule for ssh port 22 and a new TCP rule for the port configured in FoxyProxy inbound to <YOUR_PUBLIC_IP_ADDRESS>/32.

Connecting to Cluster

To ssh into your cluster, navigate to your EMR cluster in AWS console and click on ssh. This will give you the command you need to ssh into the cluster.

Accessing the Web UI

Click on Enable Web Connection in the AWS cluster console and run the ssh command for accessing the UIs through your browser. The command line argument will look like this:

```
ssh -i ~/MyKey.pem -ND <PORT_NUM> hadoop@<PUBLIC_MASTER_DNS>
```
Where `<PORT_NUM>` is the configured port in FoxyProxy (default is 8157), and `hadoop@<PUBLIC_MASTER_DNS>` is the address you use to ssh into the master cluster node. Let this run throughout your session.

**Testing your Configuration**

You should now be able to access the Hadoop UI. The Hadoop UI is located at:

```
<PUBLIC_MASTER_DNS>:8088
```

You can access Spark applications through this UI when they are running.

**Running the Mango Browser on EMR with Docker**

To run Mango Browser on EMR on top of Docker with the hg19 genome run:

```
/home/hadoop/mango-scripts/run-browser-docker.sh <SPARK_ARGS> -- /opt/cgl-docker-lib/
    →mango/example-files/hg19.genome \\n    -reads s3a://1000genomes/phase1/data/NA19685/exome_alignment/NA19685.mapped.illumina.mosaik.MXL.exome.20110411.bam
```

Navigate to `<PUBLIC_MASTER_DNS>:8081` to access the browser. In the browser, navigate to TP53, chr17-chr17:7,510,400-7,533,590 to view exome data.

The previous command runs Mango with the pre-built hg19 genome, which is in the docker container. To run the Mango browser with a different genome in Mango (ie. hg18), you will first have to create a reference genome.

To create a reference genome with docker, run:

```
/home/hadoop/mango-scripts/make-genome-docker.sh hg18 <output_directory>
```

This script will create a `.genome` file and save it to `<output_directory>` on the master host.

You can then run the Mango browser using your new genome:

```
/home/hadoop/mango-scripts/run-browser-docker.sh <SPARK_ARGS> -- <output_directory>/hg18.genome
```

The `run-browser-docker.sh` script mounts the location of your new genome file, making it accessible to the docker container.

**Note:** s3a latency slows down Mango browser. For interactive queries, you can first transfer s3a files to HDFS.

You can then run the Mango browser on HDFS files:

```
/home/hadoop/mango-scripts/run-browser-docker.sh <SPARK_ARGS> -- /opt/cgl-docker-lib/
    →mango/example-files/hg19.genome \\n    -reads hdfs:///user/hadoop/NA19685.mapped.illumina.mosaik.MXL.exome.20110411.bam
```

**Note:** The first time Docker may take a while to set up.

**Running Mango Notebook on EMR with Docker**

To run the Mango Notebook on EMR on top of Docker, run the `run-notebook-docker` script:
# Run the Notebook

```
/run-notebook.sh --master yarn --num-executors 64 --executor-memory 30g --
```

Where `<SPARK_ARGS>` are Spark specific arguments and `<NOTEBOOK_ARGS>` are Jupyter notebook specific arguments. Example Spark arguments are shown in the following example:

Note: It will take a couple minutes on startup for the Docker configuration to complete.

Navigate to `<PUBLIC_MASTER_DNS>`:8888 to access the notebook. Type in the Jupyter notebook token provided in the terminal. An example notebook for EMR can be found at `/opt/cgl-docker-lib/mango/example-files/notebooks/aws-1000genomes.ipynb`.

## Accessing files in the Mango notebook from HDFS

Mango notebook and Mango browser can also access files from HDFS on EMR. To do so, first put the files in HDFS:

```
hdfs dfs -put <my_file.bam>
```

You can then reference the file through the following code in Mango notebook:

```
ac.loadAlignments('hdfs:///user/hadoop/<my_file.bam>')
```

## 1.1.15 Running Mango Standalone

This section explains how to run the Mango browser and the Mango notebook without Docker on EMR.

### Creating a Cluster

Through the AWS command line, create a new cluster:

```
VERSION=0.0.3

aws emr create-cluster
--release-label emr-5.24.0
--name 'emr-5.24.0 Mango example'
--applications Name=Hadoop Name=Hive Name=Spark Name=JupyterHub
--ec2-attributes KeyName=<YOUR_EC2_KEY>,InstanceProfile=EMR_EC2_DefaultRole
--service-role EMR_DefaultRole
--instance-groups
  InstanceGroupType=MASTER,InstanceCount=1,InstanceType=c3.4xlarge
  InstanceGroupType=CORE,InstanceCount=4,InstanceType=c3.4xlarge
--region <your_region>
--log-uri s3://<your-s3-bucket>/emr-logs/
--bootstrap-actions
  Name='Install Mango', Path="s3://bdg-mango/install-bdg-mango-dist-emr5.sh",Args=[
    "$VERSION"
]
```

Where `$VERSION` specifies the Mango version available in the Maven central repository.

This bootstrap action will download Mango distribution code, and an example notebook file for the Mango notebook will be available at `/home/hadoop/mango-distribution-${VERSION}/notebooks/aws-1000genomes.ipynb`.
Finally, make sure you set your SPARK_HOME env:

```
export SPARK_HOME=/usr/lib/spark
```

### Running Mango Browser on EMR

To run Mango Browser on EMR on top of Docker, you will first need to configure a reference. To create a reference, see Building a Genome.

Simply run:

```
/home/hadoop/mango/bin/make_genome <GENOME_NAME> <OUTPUT_LOCATION>
```

This will save a file called `<GENOME_NAME>.genome` to your `<OUTPUT_LOCATION>`. Now that you have a reference, you can run Mango browser:

```
/home/hadoop/mango/bin/emr/run-browser-emr.sh  
  -- <path_to_genome>/hg19.genome  
  -reads s3a://1000genomes/phase1/data/NA19685/exome_alignment/NA19685.mapped.illumina.mosaik.MXL.exome.20110411.bam  
  -port 8080
```

To visualize data in the NA19685 exome, navigate to chr17:7,569,720-7,592,868. Here, you will see reads surrounding TP53.

**Note:** Pulling data from s3a has high latency, and thus slows down Mango browser. For interactive queries, you can first transfer s3a files to HDFS. The package `net.fnothaft:jsr203-s3a:0.0.2` used above is required for loading files from s3a. This is not required if you are only accessing data from HDFS.

If you have not established a web connection, set up an ssh tunnel on the master node to view the browser at port 8081.

In the browser, navigate to a TP53, chr17:7,510,400-7,533,590 with exome data to view results.

### Running Mango Notebook on EMR

To run Mango Notebook on EMR, run the mango-notebook script:

```
/home/hadoop/mango/bin/emr/run-notebook-emr.sh  
  -- <NOTEBOOK_ARGS>
```

If you have established a web connection, you will now be able to access the Mango notebook at `<PUBLIC_MASTER_DNS>:8888`.

### 1.1.16 Development notes for the Mango Browser

Please follow the Mango source installation requirements before continuing.

#### Debugging the Mango browser frontend

Mango browser uses scalatra as a web server. To interactively modify the frontend browser while running scalatra, use the “-debugFrontend” flag:
This allows scalatra to directly access ssp, css and javascript resources without packaging Mango.

- genindex
- search
CHAPTER 2

References


Python Module Index

b

bdgenomics.mango, 10
bdgenomics.mango.alignments, 10
bdgenomics.mango.coverage, 13
bdgenomics.mango.features, 13
bdgenomics.mango.pileup, 15
bdgenomics.mango.pileup.pileupViewer, 15
bdgenomics.mango.pileup.sources, 15
bdgenomics.mango.pileup.track, 21
bdgenomics.mango.variants, 14
Symbols

- __init__() (bdgenomics.mango.alignments.AlignmentSummary method), 11
- __init__() (bdgenomics.mango.alignments.FragmentDistribution method), 11
- __init__() (bdgenomics.mango.alignments.IndelDistribution method), 12
- __init__() (bdgenomics.mango.alignments.MapQDistribution method), 12
- __init__() (bdgenomics.mango.coverage.CoverageDistribution method), 13
- __init__() (bdgenomics.mango.features.FeatureSummary method), 14
- __init__() (bdgenomics.mango.pileup.sources.BamDataSource method), 16
- __init__() (bdgenomics.mango.pileup.sources.BigBedDataSource method), 18
- __init__() (bdgenomics.mango.pileup.sources.GA4GHAlignmentJson method), 18
- __init__() (bdgenomics.mango.pileup.sources.GA4GHAlignmentSource method), 20
- __init__() (bdgenomics.mango.pileup.sources.GA4GHFeatureJson method), 19
- __init__() (bdgenomics.mango.pileup.sources.GA4GHFeatureSource method), 21
- __init__() (bdgenomics.mango.pileup.sources.GA4GHVariantJson method), 19
- __init__() (bdgenomics.mango.pileup.sources.GA4GHVariantSource method), 20
- __init__() (bdgenomics.mango.pileup.sources.TwoBitDataSource method), 17
- __init__() (bdgenomics.mango.pileup.sources.VcfDataSource method), 17
- __init__() (bdgenomics.mango.pileup.track.Track method), 22
- __init__() (bdgenomics.mango.variants.VariantSummary method), 14

A
- AlignmentSummary (class in bdgenomics.mango.alignments), 10
- BamDataSource (class in bdgenomics.mango.pileup.sources), 16

B
- bdgenomics (module), 10
- bdgenomics.mango (module), 10
- bdgenomics.mango.alignments (module), 10
- bdgenomics.mango.coverage (module), 10
- bdgenomics.mango.features (module), 13
- bdgenomics.mango.pileup (module), 15
- bdgenomics.mango.pileup.pileupViewer (module), 15
- bdgenomics.mango.pileup.sources (module), 15
- bdgenomics.mango.pileup.track (module), 21
- bdgenomics.mango.variants (module), 14

C
- CoverageDistribution (class in bdgenomics.mango.coverage), 13

F
- FeatureSummary (class in bdgenomics.mango.features), 14
- FragmentDistribution (class in bdgenomics.mango.alignments), 11

G
- GA4GHAlignmentJson (class in bdgenomics.mango.pileup.sources), 18
- GA4GHAlignmentSource (class in bdgenomics.mango.pileup.sources), 18
- GA4GHFeatureJson (class in bdgenomics.mango.pileup.sources), 20
- GA4GHFeatureSource (class in bdgenomics.mango.pileup.sources), 21
<table>
<thead>
<tr>
<th>Class/Method</th>
<th>Module/Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA4GHVariantJson</td>
<td>bdgenomics.mango.pileup.sources</td>
<td>19</td>
</tr>
<tr>
<td>GA4GHVariantSource</td>
<td>bdgenomics.mango.pileup.sources</td>
<td>20</td>
</tr>
<tr>
<td>IndelDistribution</td>
<td>bdgenomics.mango.alignments</td>
<td>12</td>
</tr>
<tr>
<td>MapQDistribution</td>
<td>bdgenomics.mango.alignments</td>
<td>12</td>
</tr>
<tr>
<td>PileupViewer</td>
<td>bdgenomics.mango.pileup.pileupViewer</td>
<td>15</td>
</tr>
<tr>
<td>Track</td>
<td>bdgenomics.mango.pileup.track</td>
<td>22</td>
</tr>
<tr>
<td>track_from_json()</td>
<td>bdgenomics.mango.pileup.track</td>
<td>23</td>
</tr>
<tr>
<td>track_to_json()</td>
<td>bdgenomics.mango.pileup.track</td>
<td>23</td>
</tr>
<tr>
<td>tracks_from_json()</td>
<td>bdgenomics.mango.pileup.track</td>
<td>23</td>
</tr>
<tr>
<td>tracks_to_json()</td>
<td>bdgenomics.mango.pileup.track</td>
<td>23</td>
</tr>
<tr>
<td>TwoBitDataSource</td>
<td>bdgenomics.mango.pileup.sources</td>
<td>17</td>
</tr>
<tr>
<td>VariantSummary</td>
<td>bdgenomics.mango.variants</td>
<td>14</td>
</tr>
<tr>
<td>VcfDataSource</td>
<td>bdgenomics.mango.pileup.sources</td>
<td>16</td>
</tr>
</tbody>
</table>