1 Overview

It is possible currently to call into the GATK using Matlab (or other external tools that have tie-ins to the Java VM). There are many outstanding issues though:

- The final reduce type of the Walker is not returned to Matlab, and is only visable from the text output. It may be possible to work around this by introducing a reduce state into the walker (i.e. set a variable of type Reduce on onTraversalDone) , but this may break in future iterations of the GATK.
- You currently have to set your own parameters in the walker. We wanted to preserve the flexibility to create walkers outside of the GATK, and we haven't added support to correctly parse values out of the GATKArgumentCollection, even though the field exists.
- This feature is highly experimental, and you may run into new issues that we're unable to correctly quickly.

After saying all of that, it is possible, and we've run a couple of the basic walkers through Matlab, using the following example code, with success. We assume at this point you've at least run through the HelloWalker documentation, and can build the GATK without any problems.

2 classpath.txt

First you'll need to set your classpath.txt up to point to the GATK jar files. To find Matlab's default version of this file, type *which classpath.txt;* in Matlab. You can then copy it to your home directory and modify it there. Once you have a local copy, you'll need to add lines for each of the GATK jar's and their dependencies. The easiest solution is to include all the jar files that are build in the dist directory of your GATK checkout. For example:

Listing 1: classpath.txt example

```
...(Rest of the default classpath.txt you've copied over)...
~sting/dist/bcel-5.1.jar
~sting/dist/CombineSamAndFourProbs.jar
~sting/dist/FourBaseRecaller.jar
~sting/dist/GenomeAnalysisTK.jar
~sting/dist/log4j-1.2.15.jar
~sting/dist/picard-200903262219.jar
~sting/dist/sam-1.0.251.jar
~sting/dist/StingUtils.jar
~sting/dist/colt-1.2.0.jar
~sting/dist/commons-logging-1.1.1.jar
~sting/dist/functionalj-1.0.jar
~sting/dist/junit-4.4.jar
~sting/dist/MatchSQTagToStrand.jar
~sting/dist/Playground.jar
~sting/dist/simple-xml-2.0.4.jar
```

3 Setting GATK arguments

Now that matlab can load the classes, we need a way to tell the GATK what the default parameters should be for the run. This can be done using the GATKArgumentCollection, which contains all the required and optional arguments to the GATK.

The easiest way to load these parameters is by setting up an XML file that GATKArgumentCollection can read in. An example is provided below:

Listing 2: default.xml

```
<GATK-argument-collection>
    <sam-files class="java.util.ArrayList">
        <file>/humgen/gsa-scr1/GATK_Data/Validation_Data/index_test.bam</file>
    </sam-files>
    <strictness-level>strict</strictness-level>
    <reference-file>/seq/references/Homo_sapiens_assembly18/v0/Homo_sapiens_assembly18.fasta</reference-file>
    <analysis-name>CountReads</analysis-name>
    <enabled-threaded-IO>true</enabled-threaded-IO>
    <unsafe>true</unsafe>
    <disable-threading>false</disable-threading>
    <out-file-name>count.file</out-file-name>
    <number-of-threads>1</number-of-threads>
</GATK-argument-collection>
```

4 Example script

Now upon startup Matlab should see and load all the GATK classes. You can now setup walkers, and run them using standard Matlab syntax. Below is a sample m script to generate a read count for a given input:

Listing 3: example.m

```
import org.broadinstitute.sting.gatk.GATKArgumentCollection;
import org.broadinstitute.sting.gatk.walkers.CountReadsWalker;
import org.broadinstitute.sting.gatk.GenomeAnalysisEngine;

myWalker = CountReadsWalker();
mycoll = GATKArgumentCollection.unmarshal('default.xml');
engine = GenomeAnalysisEngine(mycoll,myWalker);
```