ASAP Reference Guide

version 1.1.4-1

Advanced Sequence Automated Pipeline (ASAP) is an application designed to help researchers easily process their sequence data into BAMs and VCFs using their local cluster or on a local linux server.

Table of Contents

1. Introduction

As next-generation sequencing becomes more affordable, researchers who wish to process the data themselves are finding the task to be both time consuming and error prone. The processing itself requires the use of many different programs, each of which has it's own set of parameters and usage requirements. Some of these programs are under rapid development and change drastically after only a few months time. As a result, processing this data requires much more than processor time. End users must configure each command executed for their particular set of data, track successes and rerun failures, ensure the integrity of their data and maintain records of which settings were used for use when writing up their results.

ASAP was designed to alleviate these issues by providing a modular system to allow users with different needs to process their data with a minimal amount of effort. In addition to minimizing human involvement, ASAP is designed to work on the researcher's local computer cluster, if one is available.

The program breaks the processing into 4 steps: Alignment, Realignment, Recalibration and Variant Calling. Processing is performed by tools produced by the research community itself, include bwa, samtools, GATK and a few others. Each of the four major steps might have subtle variations, such as how the samples are grouped during processing. Some steps, such as calling variants, even allow users to decide to use different underlying programs, such as using either samtools or GATK to call their SNPs.

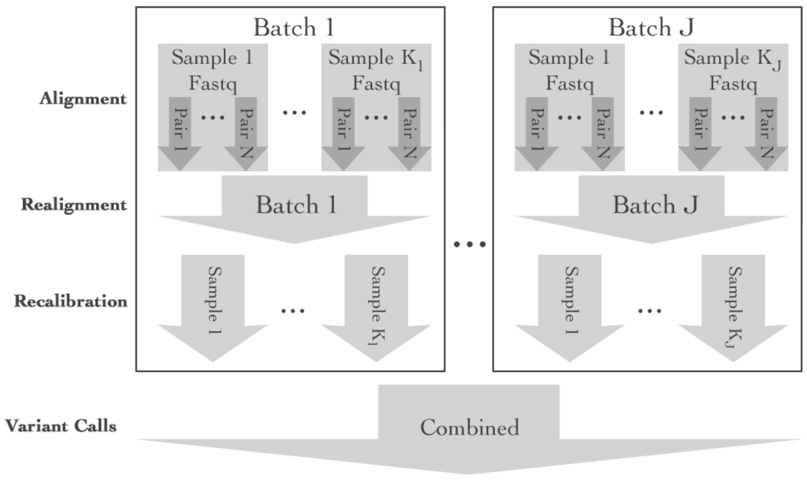

Fig. 1. Diagram of the four major steps employed by ASAP. Steps are optional, except for alignment, and grouping of samples is flexible.

With the exception of Alignment, all steps are optional. So, researchers can process their data with or without realignment and recalibration, depending on their needs. Researchers are also able to modify the behavior of the underlying applications by changing the parameters that are passed at each step. As a result, ASAP offers a highly flexible mechanism for processing sequence data. As an added benefit, all configuration settings that determine how processing is done can be saved to a single configuration file, which can be shared with other users, or reused for processing other data.

As a result of the large volume of data, processing sequence data requires many jobs to be run. It is not uncommon for some of these jobs to fail, so ASAP offers assistance in tracking down those failures. In addition to creating the scripts, users can have ASAP report the status of all jobs, reconcile job status using the cluster's job management software. When jobs fail, ASAP will write details about the job to it's error log, which ASAP provides the ability to easily print user specified numbers of lines from those files by job number, which saves time digging through large numbers of files.

While ASAP was designed for processing human fastq data into bam and vcf files, it should be suitable for use with any paired end reads as long as the user can obtain a reference genome and can be used by the various tools currently in use by the program.

2. Application Features

In addition to providing a basic pipeline to save the user time and effort in processing their sequence data, ASAP offers several features which are very important to research.

Traceability

By default, ASAP will not overwrite pre-existing scripts when a new run is performed. This allows users to go back and identify which flags were passed to any given file. Also, both error and output logs are retained in the FINAL_DIRECTORY, so users can always trace back to the commands as they were executed-this is true even for scripts that were run outside of the ASAP's job submission, since the logs themselves should always have a unique filename.

Reproducibility

Users can share their configuration files, even after removing any configuration parameters that contain local filenames. As long as the complete set of steps are present in the configuration file, and the user uses only the steps and configurations provided by ASAP itself, other users should be able to expect the same results when applied on the same data. Users that create or modify steps, templates or other functional aspects of ASAP would require sharing any modifications in order to expect complete reproducibility.

Data Integrity

ASAP uses a few different techniques to avoid issues with data integrity.

1. During execution, any system command that doesn't return 0 (standard return for successful completion) will cause an ASAP generated script to immediately fail. As such, the jobs that fall downstream of that failed command will never execute, and the data products associated with that command will not appear inside the destination directory. This ensures that no partially completed output will be accidentally used.

2. One of the very last things done by each script is move the products into place. This is done in a three steps: First, the new files are copied to a temporary directory within the destination folder. Next, the copy is then moved into place and the temporary directory is removed. This step is almost instantaneous, because the move will require only a very small change in a filesystem table. Finally, once that is completed, the original files are deleted. If the initial copy fails as a result of quota issues, or as a result of system failure or exceeding the allowed time, the final move will not be executed and the job will be reported as a failure.

Hardware Maximization

ASAP offers a number of ways in which the user can maximize their available hardware.

Multithreading

Some of the applications offer threading for some of the components of sequence data processing, but not all.

- Alignment using bwa can use threads building up the alignment coordinates, but not for the last of generating final alignments. As such, users can speed up only about 3/4 of the entire alignment process by increasing the variable BWA_THREAD_COUNT.
- Recalibration using GATK. CountCovariates is multithreaded at the time of writing this document, but TableRecalibration is not. So, only part of the process is affected by an increase in threads. Users can speed up recalibration to some extent by increasing GATK_THREADS.
- VCF Calls by GATK. We've found that using more than 2 threads to generate VCFs causes GATK to fail due to internal issues.

Users should be aware that when requesting multiple processors, they might incur a penalty for those job submissions, since it is possible that there aren't sufficient nodes with the requested number of processors available. Users should consult their cluster admins to determine what reasonable choices are for the cluster they will be using.

Splitting FASTQ files

Users can freely distribute alignments over many nodes by simply splitting the raw reads into multiple small files. Splitting a file into 10 pieces will reduce the required time to a little more than 1/10th the original runtime. There is some time associated with file splitting and subsequent merging of BAM files. However, it certainly can speed up the amount of time required to prepare whole genome data.

Chromosome Partitioning

Users are given a fair amount of control over chromosome partitioning, which is how the chromosomes are divided up and distributed across nodes during certain steps of processing. For whole genome data, it's probably best to split these up into single chromosomes for the largest chromosomes, and have the rest grouped in twos or threes. For exomes, users might want to simply have 1 or 2 partitions, since the jobs themselves won't require more than a few hours at each step. This will reduce the overhead of job management (a cluster can spend several minutes just delivering a job to the target node and launching it).

3. System Preparation

Before the user can begin processing their data, there are a few things that must be done to prepare the system. First, the user must provide a reference genome as well as ensure that each of the underlying applications that will be used are installed and accessible to the nodes that will be doing the processing.

ASAP requires that all of it's subcomponents be organized correctly in the filesystem. As such, users must not move the asap executable from it's folder. Users may create a symbolic link of the asap script and store the link in a directory if they wish, but the actual files inside the archive should not be moved.

The Reference Genome

Before a user can run ASAP, they must have a valid reference genome installed into a directory with write permission by all users who will be running ASAP.

Running Automated System Preparation

As of version 1.1, ASAP provides a step called PREP_SYSTEM, which will download outdated versions of components, install them into a local directory, configure an ASAP component manifest and finally performs each of the following steps. This can be run immediately after downloading ASAP using a command similar to:

```
asap no-cfg PREP_SYSTEM --ref-filename (path to your reference)
```

If you already have programs like bwa, samtools, etc. installed, but they aren't in your path, you should first generate a configuration file and edit the parameters associated with those program paths before running PREP_SYSTEM in order to avoid downloading and installing software that already exists on the system.

It is recommended that a valid reference genome be downloaded and installed prior to running this, though it can be run without a reference, if the user only wants to update the software. This can take between 30 minutes to an hour, depending on various issues. Users whose systems prevent long running jobs on gateway machines might have to submit this as a job under the scheduler.

Many of the files generated during system preparation that are related to the reference genome persist from run to run and need only be updated when the tools that use them are updated. Users should be aware that some older versions of these tools might not be compatible with the new files, and could crash.

Manual Preparation

Manual installation of component programs is permitted by ASAP. A list of programs and download URLs can be found in the table below. Once these tools have been compiled and installed, users will need to perform a few commands necessary to build the necessary indexes, summaries and dictionaries used throughout the processing.

The files generated at this time persist from run to run, so users should only need to do this once, and then again whenever they install newer versions of these programs.

For example, we'll assume that the reference genome used is named human_g1k_v37.fasta and is in the user's working directory. If it isn't, the user should use appropriate relative directories to ensure that the file can be found.

A Note On Compressed Reference Genomes: GATK does not support gzip compressed fasta files. Because many of the steps require GATK at this time, ASAP assumes that the reference sequence is uncompressed.

Reference genome .bwt file

This file is used by bwa during alignment, and it's absence will cause bwa to fail.

```
bwa index -a bwtsw -p human_g1k_v37.fasta human_g1k_v37.fasta
```

Reference genome .fai file

This file is used by GATK. GATK can generate it at first run, but will fail if multiple copies of GATK attempt to create it at the same time. Samtools will create it quickly without much effort.

```
samtools faidx human_g1k_v37.fasta
```

Reference genome .dict file

This file is used by GATK, as well as ASAP. It contains a quick summary of the contents of the reference itself. Users can use Picard to create this file:

```
> java -jar /path/CreateSequenceDictionary.jar R=human_g1k_v37.fasta O=human_g1k_v37.dict
```

Required Software

In addition to the reference sequence to be used during processing, users must also see to it that they have the necessary software components installed. It is recommended that the software be found in the users PATH, however, users can specify the complete path with the variable name for each application used by ASAP. As such, users can maintain different versions of these tools if there is a need without affecting ASAP's ability to use the correct one.

Application	Website	Min. Version	Min. RAM	Description	Steps This tool is Required
Ruby	http://www.ruby-lang.org/en/	1.9.2		ASAP was written in ruby and requires a relatively new version to run properly.	ALL aspects of ASAP. Please see Appendix A for instructions to install a new version of ruby as a non-root user if necessary.
bwa	http://bio-bwa.sourceforge.net/	0.5.9	3-4G	BWA is used to align the paired end reads and is currently the only aligner supported by ASAP.	Alignment
picard	http://picard.sourceforge.net/	1.6.0		Picard is a suite of tools useful for SAM and BAM manipulation.	Alignment (for MarkDuplicates) Realignment (FixMate)
samtools	http://samtools.sourceforge.net/	0.1.16		Samtools offers a number of useful functionality for manipulating sam and bam files.	All Steps that involve BAMs (Alignment - Variant Calls)

Application	Website	Min. Version	Min. RAM	Description	Steps This tool is Required
bamtools	https://github.com/pezmaster31/bamtools	2.1.0		Bamtools offers some additional functionality not available in samtools. We use it for splitting bams up by reference.	Alignment
GATK	http://www.broadinstitute.org/gsa/wiki/index.php/Downloading_the_GATK	0.5.9-r16	2G or more per task	GATK offers a large amount of functionality for manipulating as well as analysis of data within BAM files.	Realignment Recalibration Variant Calls (if GATK is selected)
bcftools vcfutils.pl	Comes with recent versions of samtools			Both come with samtools, though, the user might need to move them around a bit to get them into their path.	Variant Calls (samtools)
umake	http://www.sph.umich.edu/csg/kang/umake/download/index.html	1.0.20110706		Umake contains glf_multiples and a modified version of samtools, both of which can be used to call variants	Variant Calls (glf_multiples)

Application	Website	Min. Version	Min. RAM	Description	Steps This tool is Required
java	http://www.oracle.com/technetwork/java/javase/downloads/index.html	Modern version		GATK and picard both require java. Most machines come with java already installed, but in some cases, it might not be an official version of Java, and might not be compatible with the software. In these cases, the user will have to install a new version and ensure that it's in their path.	All steps with the possible exception of Variant Calls if the user wishes to use glfMultiples or samtools for variant calling.
ANNOVAR	http://www.openbioinformatics.org/annovar/	May 25, 2012 release		ANNOVAR will annotate SNP VCF files using various databases.	Annotation requires ANNOVAR at this time.

4. Quick Start

The following represents an example of what a typical execution pattern for ASAP might look like. For this example, it is assumed that the user has selected either MAKE_SGE or MAKE_PBS as the appropriate executor.

```
$ asap ~/asap/baseline-cfg/exome -S > cfg
```

Assuming that the user has a directory named ~/asap/baseline-cfg which contains a baseline configuration for processing exomes, would result in a new configuration file called "cfg" within the current working directory.

The user would then be expected to make some changes to this configuration file to specify project specific details. If file, exome, were didn't have details associated with system specific information, such as paths to reference genome and software applications, those would also need to be addressed. See Creating a Configuration File for more details on how to create configuration files.

Once the configuration file is ready, the user will need to add fastqs to the project using the GENERATE_BATCH command.

```
$ asap cfg GENERATE_BATCH -f fq_data -d 1 --batch_prefix PHENO1
```

In the example above, ASAP will find any fastq files within the fq_data subdirectory that match the expression defined in FQ_FORMAT. It will create 1 or more batches associated with the data found within those fq files. The data associated with this batch will be prefixed along the lines: PHENO1_D01B01. Please see GENERATE_BATCH for more information about the process, as well as the section on the configuration parameters associated with batch generation.

Once the user has loaded loaded all of the fastq files into the project, they can then begin the processing itself:

```
$ asap cfg RUN -j TRUE
```

Assuming there are no failures or other problems related to timing or system problems, that is it. The user would then wait until all of the jobs are completed.

During processing, the user can check on status using the STATUS command:

```
$ asap cfg STATUS
                                            (truncated)
1095    RUNNING /scratch/file_set-005/ALL/15_variant_call_GATK_complete_chr5.pbs
1083    COMPLETED      /scratch/file_set-005/PHENO1_D01B01/10_PHENO1_D01B01_S1_recal_chr1_2.pbs
1085    COMPLETED      /scratch/file_set-005/PHENO1_D01B01/10_PHENO1_D01B01_S1_recal_chr3_4.pbs
                                            (truncated)
Scripts In Queue  :    23
Scripts Running   :    10
Scripts Completed :    8
Scripts Failed    :    0
Orphaned BAMs     :    258

** I made all sorts of changes to the filenames here to allow the output to fit within this
window. The filenames listed as part of the STATUS command are the submitted scripts with the
full path.
```

Users can also quickly scan the contents of a log using SHOW_LOG and the job's ID:

```
$ asap cfg SHOW_LOG 1095 -L 100
```

The example above would show the last 100 lines of the error and out logs for the job associated with job ID 1095. In this case, it would be the job associated with variant calls for chromosome 5.

It should be noted that ASAP only keeps one job ID for a given "task" and for those jobs launched by ASAP. So, if the user wishes to review logs for jobs that failed after they have run ASAP to start those jobs over again, they would need to navigate to the log directory for the failed tasks themselves and manually view the logs.

Finally, should jobs fail, the user would need to first RECONCILE their job status:

```
$ asap cfg RECONCILE
```

This will cause ASAP to consult the cluster's scheduler to determine which jobs are still running, and which jobs we deleted as a result of the failure. Once that has completed, the user can simply run ASAP's RUN command once again.

```
$ asap cfg RUN -j TRUE
```

5. Running ASAP

Building Configuration

Using the steps described in the section Creating a Configuration File, the first thing a user should do is to create their baseline configuration. Once that is done, there are a handful of project specific settings one must correctly define: INTERMEDIATE_DIRECTORY, FINAL_DIRECTORY, REFERENCE_FILENAME and the Various Application Paths including GATK Settings.

In addition to the required settings, users should peruse the list of configuration parameters and review the settings that have been filled in for them before continuing. It is up to the user to describe exactly how the underlying programs process the data using ASAP's configuration settings.

Once the configuration is set, and batches have been configured, the user is ready to begin execution. The final decision is which steps should be performed. Ultimately, the more steps used, the longer processing will take, but the quality of the final data should also increase. With few exceptions, steps can be selectively used or ignored.

Splitting FASTQ Files

One way to expedite the translation of fastq data into analyzable bam and VCF files is through parallelizing the alignment phase. Users can already improve part of bwa's alignment by using threads, but bwa only uses threads for part of the alignment phase. By splitting the fastq data into smaller pieces, users can achieve a high degree of parallelization without any degradation of data quality.

Users can let ASAP split their FASTQ data using the split_fq command. This command accepts several command line/configuration arguments. This procedure is very straightforward, it simply reads the fastq file(s) and produces new files of the desired size.

Batch Generation

ASAP associates data together in three different ways: Disk, Batch and Sample. A Disk represents data that was processed together, though it might have been generated on more than one flowcell.. A batch represents data that has a common sequencing run identifier (a disk can have multiple batches) and generally represents data that was sequenced on the same flowcell. A sample represents data from a single individual within a single batch. Unless the user specifies true for ALLOW_SAMPLE_SPLIT_OVER_BATCH, ASAP will group together all reads associated with the same sample ID during a single call to GENERATE_BATCH, even if they were generated on different flowcells.

These designations are used for grouping data together during processing. In general, processing is done either by batch or by sample, depending on the user's needs. Users will issue a single *GENERATE_BATCH* command for each delivery (disk) received for all data in their analysis plan.

To perform any processing, users should add one or more batches to the ASAP application. This is done through the command GENERATE_BATCH. This command has a number of command line arguments which should apply to a specific run.

An important detail about batch information is the configuration of the FQ_FORMAT parameter and it's sister parameter, FQ_FIELD_ORDER. These two parameters are used to help ASAP identify valid fastq files, but also are used in determining the sample ID and which of the paired-end data a particular file is associated. To inform ASAP how to parse filenames correctly, they must describe a regular expression which captures the sample id and pair-end (optionally, the fastq number, if the there are more than 1 pair of fastq files for each sample). The FQ_FIELD_ORDER parameter indicates to ASAP the order those captured pieces of information will be found. For instance, the if the FQ_FORMAT where

alz-([a-z][0-9])_(1).sequence.gz and the FQ_FIELD_ORDER were SAMPLE,END then the ASAP would properly find and understand the filename, alz-100d31f_1.sequence.gz as having the sample ID 100d31f and being the first of the paired-end files. Each set of parenthesis in the format setting represents one of the fields.

It is important to note that ASAP only wants to know how to find the first end. It assumes that the other end will be 2.

Optionally, users can add NUM, to their FQ_FIELD_ORDER (along with a third set of parenthesis in their FQ_FORMAT). This is necessary only if the user's FQ data is spread out over multiple pairs. *If the user has data spread out like this, the read-end must be the rightmost 1/2 or it must be prefixed with a non-integer tag, such as R1/R2. For example, alz-2011abc_R1_0001.sequence.gz would be valid, but alz-20011abc_1_0001.sequence.gz would not.*

Step Selection

Each step is given a particular order with regard to the rest and will communicate with the preceding step during script generation. As such, users are allowed to selectively leave out steps that they don't wish to perform. Also, ASAP doesn't understand how to make proper connections for steps that effectively perform the same object but in slightly different ways, so you can't perform realignment by sample as well as by batch. As a result, users must select Alignment and the rest of the steps are optional. Steps can be set using the configuration variable, SELECTED_STEPS, as well as specified on the command line.

A list of available (and selected) steps is written to the terminal each time ASAP is run. Users can query exact spellings of step names by simply running asap with a valid configuration file or *"no-cfg"* as the only parameter.

Alignment
This step is required. This step uses BWA to align the data to a reference sequence. Alignment runtimes vary depending on how many reads are present in the fastq source file as well as how well the reads can be aligned. The output of alignment will reside in the temporary directory under the subdirectory, aligned/batch_name.

Available step names: ALIGN_PAIRED_ENDS, ALIGN_SINGLE_END

*Realignment by Batch or Sample
Realignment attempts to clean up mistaken alignments caused by the presence of insertions and deletions in a given sample's genome. Users can choose to realign by batch or sample. It is suggested to realign by batch when possible. However, when processing a very large number of exomes or many whole genomes, memory might make this unrealistic. When realigning by sample, each sample will be realigned independently.

During realignment, one or more chromosomes will be processed serially at each node. Users can control the degree of parallelization by setting the NGS parameter JOBS_PER_UNIT.

Currently, realignment is performed using GATK using two steps: RealignerTargetCreator and IndelRealigner. The output of realignment will be written to the temporary directory in a subdirectory named realigned/batch_name with each bam split by chromosomes.

Available step names: REALIGN_BATCH, REALIGN_SAMPLE

*Recalibration
Currently, recalibration uses GATK to attempt to adjust the base quality scores to be more representative of the quality of the underlying data. This is performed using GATK's walkers, CountCovariatesWalker and TableRecalibrationWalker. Recalibration is performed on one or more chromosomes at once for a given sample. Users are able to control which chromosomes are grouped together by setting the RECAL_CHROMOSOME_PARTIONING configuration setting. The more data available to GATK during this process, the more accurately the final scores will be-however, the less parallelizable it will be.

Available step Names: RECALIBRATE_SAMPLE

*Variant Calling by entire project, batch and sample
CALL_SNPS (and CALL_INDELS) calls variants from all bams inside the project. The other two options, BY_BATCH and BY_SAMPLE, pool the bams by batch or individually by sample repsectively. Users are given three applications to choose to use for variant calls: GATK, samtools and glf_multiples. The default is GATK. Users can choose an alternative template by changing the VARIANT_CALLER configuration setting.

glf_multiples Glf multiples can be used for calling with multiple samples and can potentially yield good results in a shorter amount of time. During the first time run, the glfs will be produced. However, if users are calling vcfs with additional samples, the scripts will reuse glfs where possible, which can save time.

samtools Users can use samtools/mpileup and bcftools to call variants.

ASAP allows the user to specify subregions to use for calling variants. This allows the calls on a single chromosome to be performed in a more parallel manner for large datasets.

Available Step Names: CALL_SNPS, CALL_INDELS, CALL_SNPS_BY_BATCH, CALL_INDELS_BY_BATCH, CALL_SNPS_BY_SAMPLE, CALL_INDELS_BY_SAMPLE

** GATK Realignment, Recalibration and Variant Calling Issues:*

ASAP has individual configuration settings for each of all components of GATK Realignment, Recalibration and Variant Calls. For each of these calls, GATK has slightly varied naming schemes for designating known variants which have been known to change from version to version as the software is updated and improved.

As such, each of the following flag configurations that are used should be correctly configured with the desired known variant VCFs using the relevant GATK flags: GATK_REALIGNER_FLAGS, GATK_COUNT_COV_FLAGS, GATK_VCF_FLAGS.

Failing to provide this could reduce the benefits of realignment considerably.

Moving BAM and VCFs

These steps act to finalize the highest ranking files of each type by moving them over to the final directory (by default, those files will be created inside the intermediate directory). The moves will be done in a way that safeguards against data loss/corruption, as well as replacing the original files with symbolic links in case the user wishes to reuse the data from that step in some way.

When moving VCFs containing SNP data, users can optionally allow some very basic metrics to be produced. Currently, these metrics include transition/transversion and heterozygous/homozygous alternate allele ratios. Both of these ratios are useful for identifying contamination or other fundamental issues with the data.

Available Step Names: MOVE_BAMS, MOVE_VCFS

Annotation

Users can allow ASAP to annotate the SNPs and INDELs found using the ANNOTATE_VCF step. This step uses ANNOVAR to perform the annotation, and allows the user to utilize a large portion of ANNOVAR's capabilities. When annotation is performed, the output is placed in the same directory as the VCF, inside a subdirectory named annovar.

Bam summarization is done using QPLOT which reports on a number of useful statistics for the data that has been processed.

Available step names: ANNOTATE_VCF, SUMMARIZE_BAM

System Prep

Users can have ASAP download and install most of the programs used during process. The installation does not require administrative rights, but does assume that the system has access to the World Wide Web and a valid C++ compiler.

The two components that are not installed automatically include ANNOVAR and GlfMultiples. There were some issues in automatically compiling GlfMultiples and ANNOVAR requires an email address to obtain a download link. We'll add GlfMultiples in the future, and offer the ability to install ANNOVAR if the user already has the URL.

In addition to downloading software during execution of this step, ASAP will also generate a small number of indexes, summaries and dictionaries used by the components during processing. Together, these tasks can take more than 30 minutes to complete, but should only need to be run once.

Component Manifest

Once preparation using PREP_SYSTEM is complete, a component manifest (asap/component.manifest) will exist. This file is very similar to a configuration file, but must reside in the same directory as main ASAP program itself. This file will contain system specific paths to programs and tools and is used to override any defaults in ASAP itself for paths that are found within. Users can add additional settings to this file as well and they will become part of the system's default settings-however, the file will be overwritten every time PREP_SYSTEM is run.

Importing BAM files from other sources

During processing, ASAP tracks filenames which allows it to incorporate new data as it comes in, fitting it into the appropriate place within the series of steps, executing only those steps that need to be performed. However, if a user has bam files which they want to add to an ASAP project, they can do so by using the IMPORT_BAM_FILES step. Users can import bams into an entirely new project to take advantage of ASAP's script generation to perform additional processing or just to simply perform variant calls. Users can also import bam files into a project in which other data is being processed. This allows the user to integrate data that has already been partially processed into their pipeline (or to simply add other bams into the variant calling process).

Unlike regular steps, the IMPORT_BAM_FILES step should only be called once for every "disk" of data to import.

There are a few requirements that the user should keep in mind when using IMPORT_BAM_FILES: 1) The bam files must contain only one sample, and that sample ID must be in the filename. 2) Ideally, the data will be spread out with one chromosome per bam (per sample). However, ASAP will split bams by chromosome as long as the RG tag is valid and contains the SM tag set correctly to the specified sample ID (in other words, the SM tag must also be a part of the filename and must be different from sample to sample). If ASAP doesn't have to split the bam files by chromosome, the original bam files will be used when performing subsequent steps (no extra disk space is required). However, when a split is performed, the split BAMs will reside within the intermediate directory/import_bams/ directory. This split can take quite some time, so the user should take that into account when performing the import.

When importing bam files, ASAP uses the same approach to finding the files and extracting sample_id and chromosome from the filename using regular expressions as it does during batch generation with the exception that the fields between the two are different. Please see importing bam configuration for more details on each command option associated with IMPORT_BAM_FILES.

Execution Commands

Users have a small number of execution options available. A single command must be provided or else the application will simply list configuration details.

RUN
This is the primary function for ASAP. When the user asks ASAP to run, it will produce the scripts necessary to complete the requested steps. If the user is working with a computation cluster, it will also submit the jobs with appropriate dependencies as well.

LIST_TEMPLATES
This is a very simple command that iterates over each available step and lists the templates that are available for use. This is helpful when setting up configuration, as users will need to know which template to use at any given step. This is a dynamic query, so users who create their own templates will find them listed along with those that are distributed with the program, as long as they were created properly.

STATUS
When invoking the status command, ASAP will scan the tokens and generate a report on the jobs that it finds for each status category. Categories include: Queued, Running, Completed and Failed. STATUS only applies for jobs running on a computation cluster.

RECONCILE
Users can use reconcile to synchronize tokens with the actual jobs currently running. When a script completes successfully or fails under normal circumstances, it updates the job's token appropriately. However, if the cluster's job monitor kills a job, the script is unable to properly update it's status. Also, ASAP currently doesn't know to fail dependencies when a job fails. Under both of these situations, jobs will incorrectly be thought to be either running or queued. Users can use RECONCILE to correct these issues.

SHOW_LOG job_id1,job_id2,etc
Users can use ASAP to display log information for a given set of job IDs as long as the jobs were started through ASAP itself and the job's ancestor directory structure is still fully intact. ASAP only keeps up with the last job ID associated with a single task. So, if a job fails and the user restarted ASAP to rerun those failed jobs, they would need to manually locate the log file and view it using their favorite text reader.

The logs are stored within a subdirectory of the job's script called "logs". For example, if the job's filename were /home/username/pheno1/final/pbs/fileset-001/01-scriptname.pbs the logs associated with that job would be /home/username/pheno1/final/pbs/fileset-001/logs/01-scriptname.pbs.o1234 (and a matching .e1234) where 1234 is the job id associated with the job itself.

CLEARQ
Users can have ASAP clear all jobs associated with their project using CLEARQ. This terminates each of the jobs, but only those that were launched by ASAP.

IGNORE_PRODUCT StepName
Users can indicate that products of a certain rank or higher are to be ignored, where rank is the step's position in the pipeline sequence. This is useful for situations where the data at a given step onward has become invalid, such as the combined VCF files after the user adds a new batch of fastq files for

processing. By having ASAP ignore MoveBams onward, the new fastq files will be processed and, upon completion of the final bam processing step, the new files will be properly integrated into the system for tasks like SNP calls and annotation. StepName is the actual name of the step whose rank is to be the lowest rank ignored and must be in CamelCase style notation. If there are sister Steps with the same rank (such as CallSnps and CallIndels), all steps of that rank and higher will be ignored.

When performing this function, ASAP will delete any symbolic links it encounters. During MOVE_BAMS and MOVE_VCFS, symbolic links are created to prevent unnecessary copying, but still allow each step to be able to find their own files should they be asked for them. This shouldn't pose any problems, and will only affect data in the INTERMEDIATE directory.

If data from the INTERMEDIATE directory has been deleted, and subsequent steps ignored, the system will attempt to reprocess from the highest rank that that has valid data associated with it, which might be the very start. Therefore, if users expect to get additional fastq data and wish to call SNPs and INDELs on the entire dataset as opposed to calling by batch or sample, they should either avoid deleting INTERMEDIATE/recalibrated or should avoid ignoring MoveBams. INTERMEDIATE/recalibrated should contain only symbolic links, so it should only affect file count quota and not disk space.

6. File and Directory Structure

ASAP produces a lot of files and uses some potentially complex directory structure. However, this allows it to be flexible. Below are some of the most important details regarding file structure. All of the paths listed below are relative to either the FINAL_DIRECTORY path or the INTERMEDIATE_PATH.

FINAL_DIRECTORY Contents

Directory/Filename	Description
serial-exec	Each executor writes it's scripts into it's own directory within the FINAL_DIRECTORY. This is root of all serial based scripts.
serial-exec/file_set-XXX	Each time you perform a RUN with ASAP, a new file_set is generated, incrementing the numerical portion of the filename by 1 more than the highest present. This directory contains all scripts
serial-exec/file_set-XXX/auto-serial-masther.sh	This is the master script that controls serial execution. Users can use this script if they want completely serialized processing. However, if they wish to run multiple jobs simultaneously on the same system, they will need to do that manually.
serial-exec/file_set-XXX/tool_details.log	This file indicates the versions of the various applications found at the time the ASAP created the scripts. This is logged so that users can have an idea which versions of each tool might have been used at the time of execution. Please note, however, that this is done at the time of script production, not execution. So, if programs or paths are changed prior to a script's execution, the log is misleading.
bam/BATCH_DXXBXX	All bams are stored together by batch.

Directory/Filename	Description
vcf/snps_gatk/ vcf/snps_samtools/ vcf/snps_glf_multiples/	SNPs are stored in a more complicated manner. Because users might call SNPs using different templates, ASAP uses the template name for the directory immediately inside the vcf directory. Then, depending on whether the SNPs were called all together, by batch or by sample, the VCFs will be stored in different paths.
vcf/indels_gatk/ vcf/indels_samtools/ vcf/indels_glf_multiples/	Indels are stored in the same structure as SNPs-the base folder within vcf is determined by the template used to perform extraction and the contents within depend on the type of grouping performed.

INTERMEDIATE_DIRECTORY Contents

In general, users probably won't need to bother with the files inside these directories. However, the files do persist as long as the user permits them to.

Directory/Filename	Description
aligned/BATCH_DXXBXX	Output of the alignment. Aligned files should all be compressed (bam) and indexed (.bam.bai) 1 for each chromosome for each fastq file
realigned/???	Data to be realigned can be grouped by batch or by sample. This grouping determines what the subdirectory will be called. Bams will be indexed and will be 1 chromosome per batch or 1 chromosome per sample based on which grouping was used.
recalibrated/BATCH_DXXBXX/sample_id/	Recalibrated bams will be indexed and be 1 per chromosome per sample

Directory/Filename	Description
vcf/template/ALL_BATCHES (or BATCH_DXXBXX)	VCF files. 1 per subregion per chromosome per sample (or batch or ALL depending on the grouping selected)

7. Creating a Configuration File

ASAP's configuration file is formatted in a very simple way. Anything following a "#" sign is considered to be a comment and is ignored. The parameter name is specified as the first thing on the line followed by the value to be assigned, separated by a space or a tab. The parameter's name should be all upper case. With only a few exceptions, the value should not contain spaces, tabs or any other non-visible character. The exception to this are the flags to be passed to the underlying programs. These are expected to have spaces in them. Because of these rules, filenames specified in the configuration file should not contain spaces.

Sources for Configuration Settings

ASAP obtains configuration parameters from three sources: environment, configuration file and in the form of command-line arguments in that order. As a result, a parameter that is specified on the command-line overwrites whatever might have been specified in one of the other sources.

Parameters From Environment

In general, users will not want to add ASAP specific details to their shell environments, however, there are times when users might want to use environment variables to pass information to ASAP, such as when running ASAP as part of a larger script.

```
$ qsub myjob
12345.scheduler-name
$
```

To assign parameters through the environment, users can use the parameter's name as the environment variable. As such, users can assign any configuration parameter using environment flags. For example, in the command

```
> REFERENCE_FILENAME=/path/to/reference asap exome.cfg -S > my.cfg
```

we are setting the default value for REFERENCE_FILENAME to /path/to/reference. This can be useful for setting variables which contain filenames during configuration file generation. It is important to note, however, that in the case above, if REFERENCE_FILENAME were present in the file, exome.cfg, the setting found in the file would be used, rather than the value from the environment.

Parameters From the Command Line

Some parameters can also be specified through command-line arguments. for example, we can set the reference path using the command-line argument, --ref-path, and using the command

```
> REFERENCE_FILENAME=/path/to/reference asap my.cfg --ref-path /this/will/be/used/instead
```

would override both the environment and configuration settings.

Creating a New Configuration File

Users can generate a new configuration file based on the current settings (either from environment, configuration file, command-line or any combination of the three) by using the -S flag. The configuration is written to the standard out and can be redirected to a file. The command

```
> asap examples/example.cfg --ref-filename /path/to/reference.fasta -S > new_configuration.cfg
```

would first load any configuration data from the environment (we don't see any shown explicitly here, but there could be some that were exported), then the file, examples/example.cfg and then the --ref-filename and write those parameters along with any missing default parameters to the file, new_configuration.cfg.

It is recommended for teams to keep a baseline configuration file and extract new project configurations for each new set of data that they wish to process. In addition to giving all users for the team a common baseline for functionality, it also ensures that any default settings that aren't present in the baseline configuration file be present in the newly generated one. Users can then browse their newly generated configuration file to see if there are appropriate fields to fill in. This can be helpful if the application has been updated since the original *template configuration* was generated, since any new parameters would then be written to the new file.

Using this approach to generating new configuration files for new projects offers some important benefits over simply copying configuration files. First, if the original configuration file were generated by an older version of ASAP, the newly generated file will reflect any new parameters that have been added, ensuring that the user doesn't miss new options that have become available. Next, by filtering out project specific settings in the original configuration file, like the final and intermediate paths, the users can be sure that those variables will not accidentally have invalid settings (in fact, if a missing setting is required, ASAP will simply report those missing settings as an error and halt immediately). This helps eliminate some types of errors such as importing fastq files into an unrelated project because the user forgot to correct the FINAL_DIRECTORY parameter. Lastly, users can clean these configurations of all things specific to their system (such as paths to the reference), and post them on a website to share with other researchers as suggested "best practice" configurations.

GATK 2.x users

There were some changes made to Recalibration which makes the scripts produced by ASAP incompatible between the 2.x and previous versions. As such, users that wish to use GATK 2.x must either manually make a few changes to their configuration to work with GATK2, or they should instruct ASAP to use the GATK2 specific settings when generating their configuration.

To have ASAP make the necessary changes during configuration generation, set the environment variable, GATK2=TRUE as shown in the example below:

```
> GATK2=TRUE asap baseline/example -S --batch-prefix EXAMPLE --ref-filename ref/chr21.fasta \
 --dbsnp-vcf ref/1K_chr21.vcf --picard-path /full/path \
 --gatk-path /full/path/to/jar > example.conf
```

(You don't have to specify picard or gatk's paths if you used PREP_SYSTEM).

To make the changes manually, users will have to change the following three configuration parameters:

Change RECALIBRATION_TEMPLATE to gatk2 (it should have said gatk).

Correct the flags associated with GATK_COUNT_COV_FLAGS and GATK_RECAL_FLAGS so that they are suitable for the GATK modules BaseReecalibrator and PrintReads respectively. ASAP provides the following as default settings:

Step Configurations

In ASAP, a step is a group of commands that will be executed together, generally to perform one of the primary tasks such as alignment or recalibration. In addition to the primary four, there are a few additional steps that move and combine files into the final directory to make analysis easier.

Steps are activated at the launch of ASAP. Once they are active, they will then tell ASAP about their configuration parameters. As a result of this, users should always specify all steps they plan to use during processing when creating their configuration file. This ensures that all relevant configuration parameters are known before the user begins processing data. In some cases, configuration settings can be shared between 2 or more steps. In these cases, they will have an identical spelling for the parameter itself and the setting will work for either. There will only be one entry for such identical settings.

Steps can be passed to ASAP using all of the normal ways, with one exception, the command-line. When one or more steps appear on the command-line, ASAP will ignore the contents of the SELECTED_STEPS parameter. So, if a user doesn't want to use the default steps, and don't have an example configuration containing the steps they want to use when creating their new configuration file, they should list all of the steps on the command-line, separated only by spaces. The following command

```
> asap examples/example.cfg ALIGNMENT_NO_SPLT CALL_SNPS_BY_SAMPLE > new_configuration.cfg
```

would generate a configuration which contains only configuration parameters associated with the steps: ALIGNMENT_NO_SPLIT, CALL_SNP_BY_SAMPLE, regardless of how many steps were present in the configuration file, examples/example.cfg.

8. Shared Configurations

Baseline Configurations

ASAP's configuration file mechanics are designed to allow users to reuse partial configurations to ease the burden of setting up projects. For example, a team that processes data for multiple different projects might want to use the same basic steps and configuration for processing, but they would need to keep the different intermediate and final products in separate directories. This can easily be achieved by saving a copy of a functional configuration in a common working area, removing anything specific to the given project, such as INTERMEDIATE_DIRECTORY, FINAL_DIRECTORY and BATCH_PREFIX entries from the new file. In future executions of ASAP, users can use this new file, along with the -S option to generate new configuration files to which they would tweak to make appropriate changes particular to the new project.

Sharing with Colleagues

Shared configurations are incomplete and should never be used as actual configurations except when using ASAP to generate a new configuration (-S option).

The same approach can be used to share with colleagues as well. Any variable that is missing from a configuration file is populated by the application's default values. So, removing any settings that refer to path specific options can safely be removed, and would be populated by the recipient. The one loophole here, is for settings, such as several of those for GATK where the a path is required for one of the parameters. This makes sharing these settings a bit trickier, since the end user must edit those paths to correspond to their system, but otherwise, works fine.

Component Manifest

The component manifest optionally exists in the same directory as the actual ASAP script file and is intended for system specific settings such as the paths to applications. When a user runs PREP_SYSTEM, ASAP will write a component manifest file into the correct place using the paths where the software was run. When this happens, only those variables associated with application paths are written to the manifest. However, any variable found in a configuration file can be added and the file can be edited by a user with write permission to that file. Anything found inside the manifest overrides the normal defaults for ASAP, allowing users to define systemwide settings for every file based setting.

The order of precedence, from lowest to highest is:

Environment, ASAP default, component manifest, configuration, command-line settings

9. Configuration Parameters

The following represent each of the configuration parameters and a short description of what they are used for. With the exception of **FLAGS**, the parameters accept only a single value separated from the first word on the line by whitespace (either space(s) or tab). Users can add a '#' character to have the application ignore any lines. Any configuration setting not present (or commented out with a '#') will use the default value.

Configuration parameters for all active steps will be written to standard out when the user provides the command line flag, -S. In addition to the current default values (those reflect the current state, which could have been adjusted by the application of a configuration file, environment or command line), there should be basic comments describing the purpose of the parameter. Users are recommended to check the actual list in case this document is outdated from the application. Sample configurations will only contain parameters associated with the current set of SELECTED_STEPS.

The parameter listings below indicate command-line arguments in parenthesis when available. In some cases, the only way to change a setting is through command-line arguments. In these cases, the flags are listed without a matching variable name.

A note about (nothing). When a parameter defaults to nothing, it results in the variable being commented out (or sometimes to an empty string). For many settings, this is a valid setting. However, in some cases, this is intentionally an erroneous setting and will result in an error during execution if the user doesn't change it. This is often used for required paths, such as FINAL_DIRECTORY, where ASAP can't make a reasonable guess as to what is reasonable. For these configuration parameters, ASAP will use (nothing, but required).

Special Parameter Types

The following types appear below describing the type of parameter values to be provided. The types below are highly specialized in their format and aren't necessarily clear by their name alone.

TRUE/FALSE

These parameters represent boolean values. Technically, only TRUE is tested for (so if the value isn't TRUE, it is considered to be false). These settings are not case sensitive.

regex

Regular expressions are used to describe formatting for certain filenames and help capture certain elements from the filename, such as sample ID or file number. Regular expressions can be tricky to learn, and it is recommended the user consider spending a little time reading about them before making changes to these variables. If users wish to experiment with a regular expression so that they can be sure theirs correctly captures the relevant information, they can use this website.

name-or-path-to-file

Parameters specified as such can either be the standard application name, if the application will be found in the user's PATH during job execution. In some clusters, the PATH associated with a job may differ from the PATH of a user's login shell. In this situation, the parameter should contain the full path of the application as the node will need to use to properly execute the application. This also allows users to specify a particular version of the application to be used than the version they might be using for other purposes.

chromosome-partition

A chromosome partition simply describes how the step will group chromosomes when possible. In some cases, the grouping is done randomly. However, when it's relevant for accuracy or efficiency, ASAP grants the user the opportunity to explicitly describe how the chromosomes will be grouped.

A chromosome partition is a comma separated list of chromosome groupings. Such a grouping can contain one or more chromosomes combined by '+'. An example might look as follows: 1,2,3,4,5,6,7,8,9,10,11,12,X,13+14+15,16+17+18,19+20+21+22+Y+MT. Chromosomes 1-12 and X would be treated as groups and treated independently of other chromosomes. 13-15 would be treated together, as would 16-18 and 19-22 (along with Y and MT). The names used should be the same as those found in the reference sequence.

It is important to note that spaces are not allowed in chromosome-partition values.

template-selection

In ASAP, the term template refers to the templated code that will be translated into the final scripts that perform the data processing. In some cases, a step might have multiple templates, or alternative ways to perform the same task. These "templates" are selected by the user by a simple string passed through the configuration or on the command-line. These template names must match the templates currently available for the given step. Template options that are delivered with ASAP will be listed in the manual.

flags

These are ASAP configuration parameters which specify flags the user wishes passed on to a given program execution during processing. These parameters should contain all of the command line arguments the user wishes to pass through to the program doing the actual processing (such as bwa or GATK), except for those which ASAP must know for other reasons. For instance, some GATK commands allow for using multiple threads, however, ASAP must know how many threads the user wishes to use. As such, the user should specify this by setting the GATK_THREADS parameter so that the ASAP can properly estimate execution times as well as make proper resource requirements when submitting jobs to the scheduler.

These variables can be quite long and can contain many spaces. However, they should not contain a return character.

Currently, ASAP does not analyze the settings contained within these parameters. So, users should be careful to avoid providing those flags which ASAP must also provide (such as GATK_THREADS or GATK's -T walker-name flags). These will likely result in an error during the job's execution.

By default, ASAP comes with reasonable settings for these, but it is highly recommended that the end user familiarize themselves with those settings to ensure that they understand what is being done.

General Parameters

SELECTED_EXECUTOR (-e) string
Users can specify an alternative executor in their configuration file. This would allow users to utilize other cluster technology, such as the Sun Grid Engine, or to run their scripts in a serial fashion. Current executors provided include: MAKE_PBS, MAKE_SGE, MAKE_SERIAL.

Default value is MAKE_PBS

JOBS_PER_UNIT integer
During job production, a unit is a set of tasks required to prepare the requested data for the subsequent step. In some cases, this might be something like "realign batches X,Y and Z, chromosomes 1,2,3,4,5,6,7,8,9,10". Based on the setting for JOBS_PER_UNIT, the realignment step would then decide how many different jobs to create to satisfy that request. In the example above, the actual number of jobs produced would be between 3 and 30 depending on the setting for the JOBS_PER_UNIT.

Default value: 25

GROUP_OWNER group-name
This specifies a unix-group name to which the result files of the application will be *owned*. This is helpful when users other than the person doing the processing needs to access the files.

Default Value: (nothing)

BATCH_PREFIX (--batch-prefix) string
Specifies the prefix applied to each bath name when generating new batches. This will be used in various directory and file naming schemes.

Default Value: BATCH_NAME

SELECTED_STEPS comma-separated-list
Comma separated list of processing steps to be activated. Users can also provide step selections on the command line by simply adding the desired step in capital/underscore notation (i.e. ALIGNMENT_SPLIT). If one or more step appears on the command line, steps in the configuration file are ignored.

Default Value:
ALIGNMENT_SPLIT,REALIGN_BATCH,RECALIBRATE_SAMPLE,MOVE_BAMS,CALL_SNPS,MOVE_VCF

CHROMOSOME_LIST comma-separated-list
Users should specify how the chromosomes are named in their reference. By default, the naming used is the same as the reference genome used by the 1000 Genomes Project: 1-22,X,Y,MT.

Default Value: 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,X,Y,MT

IGNORE_UNKNOWN_ALIGNMENTS TRUE/FALSE
By default, any reads that do not align to one of the chromosomes listed in CHROMOSOME_LIST will be stored in an extra file, which will have unmapped as part of it's filename. This file will also include any unaligned data as well. If users don't want to keep this data, they can set this value to TRUE.

Default Value: FALSE

FQ_FORMAT regex
This variable is used during batch generation and is used to help ASAP do two things: 1) it helps ASAP parse the filename for the fields listed in FQ_FIELD_ORDER and 2) helps ASAP discover the path to the first end of paired end data.

Once the first file for each pair is found, ASAP will then infer the filename associated with it's mate based on the expression. In order for ASAP to do this, the user must provide a regular expression that "captures" each of the components specified in the FQ_FIELD_ORDER setting. Those fields include: Sample ID, paired end (1 or 2) and fastq file number. The fastq file number is optional, and is only likely to be present if the files were split by ASAP for faster aligning. This number represents which of the N pieces of the original fastq this particular file is (i.e. if the file were split into 20 smaller fastq files, that number would range from 001 to 020.)

An example of defining a regular expression might be as follows: If the user were to have files with names like: ABC_M1234_R1_sequence.gz, where M1234 was the sample ID, R1 indicated it was the first of the pair, the regular expression might look like: [A-Z]+_(M[0-9]+)_(R1)_sequence.gz and the FQ_FIELD_ORDER would be SAMPLE,END.

In the expression described above, [A-Z]+_ means it should ignore one or more contiguous capital letters, followed by one "_" character. Next, it will capture an "M" followed by 1 or more digits (0-9). This will be identified as the sample ID because it is the first value captured, and that corresponds to the first field in the FQ_FIELD_ORDER, SAMPLE. Next, there is a lone "_" character which will be ignored followed by (R1). This will be captured as the END, since it's the second item in FQ_FIELD_ORDER shown above. The rest of the filename will be ignored and is assumed that it will be the same for every other file.

It is important to note that the END portion of the expression should never refer to the reverse end of the paired end. ASAP will find the matching end by assuming it's mate will have a 2 in place of the 1 (or, as in the case above, an R2).

Another potential error associated with fastq file names that should be kept in mind is as follows: When the designation for which paired end is simply a numerical value of a 1 or a 2, that number MUST be the right most number 1 (and 2). Currently, if the designation includes an "R" (upper or lower case), that R must be preceded by an underscore. These restrictions are necessary for allowing ASAP to correctly identify which files are associated together.

Default Value: ([a-zA-Z0-9]*)_[a-zA-Z0-9_]*(R1)_([0-9]+).fastq.gz

FQ_FIELD_ORDER comma-separated-list
The fields associated with the filename. Each of these fields represents one of the *captured* elements of from the regular expression defined in FQ_FORMAT. In general, SAMPLE and END are required. However, if the fastq files were split using ASAP, those will also have the FILENUM portion, which indicates which of N files a given fastq file is.

Default Value: SAMPLE,END,FILENUM

RUBY_PATH path-to-ruby
This is an optional setting for users whose default ruby isn't 1.9.2 or newer. If this is the case, the user should provide their alternate path to the ruby interpreter (named ruby). If this is not defined (or defined but without a value), ASAP will use the unix "which" command to determine what the path to the default interpreter.

Users can install ruby if necessary, even if they lack administrator rights. See Installing Ruby for instructions.

Default Value: (nothing)

Cluster Based Parameters

The following parameters are specific to the executors, MAKE_PBS and MAKE_SGE.

JOB_ACCOUNT (--job-account) (-a) account-name
When running on a cluster, users are often allocated usage limits based on unix group "accounts". An individual user might be permitted to use more than a single account. ASAP allows users to run their jobs under an alternate unix group by changing this parameter. If left blank, the default group is used.

Default Value: (nothing)

EMAIL_ADDRESS user-email-address
If the system supports it, users can receive emails indicating that the job: Aborted, Begun and Ended.

If left blank, none of the email settings will be used when submitting jobs.

Default Value: (nothing)

EMAIL_ON_ERROR_ONLY TRUE/FALSE
If an email address is given, this will suppress the begin and successful termination messages when set to TRUE.

Default Value: TRUE

TIME_CALIBRATION float
Adjustment for time. Users can increase this number if their jobs are being killed for running too long. The default value of 1.0 represents no adjustment. A value of 2 would double the amount of time requested for jobs submitted.

Default Value: 1.0

PREFERRED_TEMPORARY_STORAGE (--preferred-temp) (-t) path
Users should set these to paths that will be valid on the node(s) on which the jobs will be run. It is recommended that this setting be a local disk on the node itself (such as /tmp). However, it can be any properly mounted path.

Default Value: /tmp

ALTERNATE_TEMPORARY_STORAGE (--alternate-temp) (-a) path
This is a secondary filesystem for which the application will write intermediate data during processing on cluster jobs. This is used if there is insufficient diskspace on PREFERRED_TEMPORARY_STORAGE.

Default Value: /scratch

JOBID_SUFFIX string
This value is used to tell asap if there will be anything added to the actual job id when it queries the environment variable, PBS_JOBID. This is necessary for finding log files, since the contents of that variable are appended to each log that is produced during execution on a PBS based cluster.

If you are unsure about what this might be, simply submit a job to the cluster. If is successful, it should turn either a number, or a number followed by what looks like a computer's network address. Anything that follows the number is considered to be the *JOBID_SUFFIX*. For example, when a job is submitted using a particular cluster, the following might be returned:

In this case, the JOBID_SUFFIX should be set to .scheduler-name (including the preceding ".").

Default Value: (nothing)

--job-submit (-j) TRUE/FALSE
Specify whether jobs should be submitted or only written to file. Currently, only those jobs submitted by the system are properly managed with job dependencies.

Default Value: FALSE

--overwrite-fileset (-o) TRUE/FALSE
Specify whether jobs should reuse the last pbs(or sge)/file-set directory. This can be used to minimize clutter, but if the user changes application flags, then it's possible that the tracking will be incorrect for jobs that completed in the particular directory that were later overwritten.

Default Value: FALSE

Sun Grid Engine Details

The following configuration options are only valid for use with the MAKE_SGE executor.

QUEUE_NAME string
Allow the users to specify which queue to submit their jobs. If left blank, the default queue will be used (none will be included in the job script). The possible values for this will be cluster specific. If the user doesn't need to specify a particular queue during normal submissions, they should leave this blank.

Default Value: (nothing)

PARALLEL_ENV string
Specifies which Parallel Environment (PE) to use when submitting the jobs. This setting is very cluster specific. However, it is important that the users select a PE that allows multiple cores, or setting their BWA_THREAD_COUNT and GATK_THREADS settings to 1.

If a user is not sure which PEs are available, they should consult their cluster administrator for assistance.

Default Value: threaded

PRODUCT_PERMISSIONS integer
Permission settings to be used on the files produced during processing. This is the same permission value that would be passed to chown (i.e. 666 for all open).

Default Value: 640

MEMORY_FLAG string
The actual name of the flag to be used when setting the memory request during job submission.

Default Value: mem_free

Serial Execution Parameters

The serial executor also uses the --job-submit (-j) command line option.

SCRIPT_NAME string
Specify the name of the master script which executes each of the scripts generated in a serial fashion.

Default Value: auto-serial-master.sh

KEEP_TEMPORARY_FILES TRUE/FALSE
When true, the temporary working directory isn't deleted upon completion. This is useful if there is a problem and the user wishes to examining files that are failing.

Default Value: FALSE

VERBOSE_ERROR (--verbose-error) (-e) TRUE/FALSE
When true, additional details will be displayed in the error log when intercepting an error from one of the pipeline programs. This is useful only when trying to to identify the cause of failures.

Default Value: FALSE

General Path Details

FINAL_DIRECTORY full-directory-path
This setting specifies where the *permanent* files are stored. Such files include, application configuration, job tokens, PBS scripts and logs as well as the final data such as VCF and bam files associated with the steps MOVE_BAMS and MOVE_VCF.

Default Value: (nothing but required)

INTERMEDIATE_DIRECTORY full-directory-path
This setting specifies where the data that is produced by one step can be found by subsequent steps. It is recommended that users have at least 4x available space in their quota on this path as their total fastq size.

Default Value: (nothing but required)

REFERENCE_FILENAME filename-of-reference-sequence
This can be either a filename, or a filename with full path.

Default Value: (nothing but required)

REFERENCE_PATH path-to-reference-sequence
This specifies the fully qualified path to the reference sequence to be used. This can be commented out if the REFERENCE_FILENAME contains the full path.

Default Value: (nothing)

JAVA_TMP_DIR full-directory-path
If set, this directory will be passed to java as -Djava.io.tmpdir=$JAVA_TMP_DIR. This is only necessary if the default directory, /tmp, is not writable by users on all nodes. By default, this value is nil and isn't passed to any java applications.

If left blank, java will use it's internal default path.

Default Value: (nothing)

GATK Details (Versions 1.6.x and 2.x)

The following GATK settings are general GATK settings. For settings specific to a particular step, please see the settings for that step as well as the documentation for GATK itself.

GATK_THREADS integer
Specifies the number of threads to pass to those GATK steps that can use it. This will increase the number of processors requested, and, if too high, could affect how quickly the job will be started.

Default Value: 2

GATK_JAVA_MEM integer
Specifies how gigabytes to assign to java when executing GATK. This will affect all steps that use gatk.

Default Value: 4

GATK_PATH path-to-gatk-jar-file
Specifies the full path to the GenomeAnalysisToolKit.jar file.

Default Value: (nothing)

GATK_COMBINE_VAR_FLAGS flags
The flags passed to GATK during merging vcf files which contain different samples.

Default Value: -genotypeMergeOptions UNSORTED

A Note on GATK 2.x

With the release of GATK 2.0, GATK made some big changes, including offering a Lite version which has only a subset of the tools under development. ASAP users who wish to use GATK 2 must use the full version, which requires subscribing to their forums.

Because of this requirement, ASAP is unable to automatically install GATK2.x.

Please download it manually at this location: http://www.broadinstitute.org/gatk/auth?package=GATK and follow the instructions below for specifying the correct path for GATK.

Users will need to manually change the GATK_PATH to point to the GATK jar file.

~~~~~

ASAP can switch some default settings that are appropriate for GATK2.x if GATK2=TRUE in the user's environment when generating a new configuration file. **NOTE** This only works if the relevant settings are not present in either the component.manifest or in the current configuration file (if not no-cfg).

Software Names and Paths

For all application paths, if the application resides in the user's PATH, the name of the application is sufficient. However, if the user wishes to use something that isn't in their path, or a different version than
~~~~~

one that lies in the path, they should provide the full path to the program. Also, if the user's cluster doesn't properly load the PATH variable, they might have to always provide the complete path.

BWA_APPLICATION name-or-path-to-file
Specifies how to call bwa.

Default Value: bwa

BAMTOOLS name-or-path-to-file
Specifies how to call bamtools.

Default Value: bamtools

SAMTOOLS name-or-path-to-file
Specifies how to call samtools.

Default Value: samtools

PICARD_PATH path
Specifies the full path to where the picard suite tools are found. This is only the path, and not a full path to a particular file.

Default Value: (nothing)

UMAKE_SAMTOOLS name-or-path-to-file
Umake provides glfMultiples, which depends on a special version of samtools to generate the glfs. This parameter specifies the name or full path to this application.

Default Value: samtools-hybrid

GLF_MULTIPLES name-or-path-to-file
Specifies the name or path to the glfMultiples application.

Default Value: glfMultiples

VCFUTILS name-or-path-to-file
Specifies the name or full path to the vcfutils.pl script.

Default Value: vcfutils.pl

BCFTOOLS name-or-path-to-file
Specifies the name or full path to the bcftools application.

Default Value: bcftools

Split FASTQ

The following parameters are specific to splitting the FASTQ files.

MAX_READ_COUNT (--max-read-count) integer
Specifies the maximum number of reads in the split files.

Default Value: 16000000

FQ_SOURCE_REGEX (--fq-source-regex) regex
This specifies the regular expression used to extract the various components of the filename.

This setting is very similar to the FQ_FORMAT with the exception that this is the format used to find the fastq files to be split.

Default Value: ([A-Za-z0-9]*)_[rR]*([12])[a-z.A-Z]*.gz

FQ_SOURCE_FIELD_ORDER (--fq-source-field-order) comma-separated-list
This specifies the order in which the various components of the filename are found. Please see FQ_FIELD_ORDER for more details.

Default Value: SAMPLE,END,FILENUM

MAX_JOB_COUNT (--max-job-count) integer
Specifies the maximum number of independent jobs to be launched during the split step.

Default Value: 2

--fq-file-list-filename filename
File containing the fastq files to be split. There should be one file on each line.

Default Value: (nothing)

Automatic System Preparation Settings

The following settings are used to configure the tool, PREP_SYSTEM.

DOWNLOAD_COMPONENT_BIN (--tool-bin) path
Specifies the path to where the programs will be found. In reality, PREP_SYSTEM only makes symbolic links from the DOWNLOAD_COMPONENT_SRC directory, but it is the links found inside DOWNLOAD_COMPONENT_BIN that are written to the component manifest.

DOWNLOAD_COMPONENT_SRC (--tool-src) path
Specifies the path to the directory where sources are extracted and programs compiled. PREP_SYSTEM will link any binaries from within this directory into DOWNLOAD_COMPONENT_BIN, so users should not delete this directory once preparation is complete.

DOWNLOAD_COMPONENT_ANN_URL (--annovar-url) url
In order to avoid bypassing ANNOVAR's request for email addresses, ASAP does not include the URL for downloading ANNOVAR software. However, the user can provide this URL during execution of PREP_SYSTEM and it will be correctly downloaded and tied into the component manifest.

Batch Generation

As of version 1.1.3, users can choose between using regular expressions to find their FASTQ files and using a tab separated text file to specify which FASTQ files are associated with which samples.

ALLOW_SAMPLE_SPLIT_OVER_BATCH TRUE/FALSE
When false, a sample that was processed on more than one sequencing run would be treated as a single sample the reads from these runs will be merged. Each will retain the correct sample ID, however, different "read group" tags will be applied during alignment, which might result in being split out differently at later processing steps.

Default Value: FALSE

READ_LENGTHS comma-separated-list-of-integers
Users can specify a list of valid read lengths. ASAP will halt if it encounters reads that don't match one of the entries in the list. Users can set the list to be empty to accept any read length.

Default Value: (nothing)

ALIGN_CALIBRATION float
The average alignment time per read for exome data appears to take less time than for whole genome data. As such, users can adjust the expected time required for alignment by providing a multiplier (i.e. 2.0 means to allow for twice as long as expected). The default estimation is based on exome data and we recommend setting this value to 5.0 or 6.0 for whole genome to be safe.

Default Value: 1.0

Batch Generation - Regular Expression Search

--fq-path (-f) path-to-fastqs
This specifies where the fastq files can be found. The application will scan subdirectories. Any fastq file that matches the regular expression definition will be incorporated into the "batch". However, the data will be further divided according to read group information inside the fastq data itself.

Default Value: (nothing)

--disk-number (-d) integer
It is necessary to specify a unique number for each disk being processed into batches.

Default Value: 1

Batch Generation - Flat File Specification

--fq-file-list file
Users specify the name of the file containing the sample ID and complete filename for each FASTQ on a given disk. Sample IDs should not have spaces, and the two fields should appear together on each line separated by a single tab.

Setting this causes the GENERATE_BATCH command to ignore regular expression settings.

Default Value: (nothing)

Alignment Settings

ALIGNMENT_TEMPLATE template-selection
Specify the template to be used during alignment. Currently, ASAP comes with only one option for this, bwa.

Default Value: bwa

Alignment Settings (BWA)

BWA_THREAD_COUNT integer
Specify how many threads to use during the aln phases of bwa. Current version of bwa does not provide threaded support for it's last phase, sampe.

Default Value: 2

BWA_ALN_ADDITIONAL_PARAMS flags
Users can specify all additional parameters to be passed to bwa during the ALN call.Users should consult the bwa documentation to determine which settings are appropriate for their needs.

Default Value: -q 15

BWA_SAMPE_ADDITIONAL_PARAMS flags
The flags to be passed to bwa during the SAMPE call. Users should consult the bwa documentation to determine which settings are appropriate for their needs.

Default Value: -a 500 -o 100000 -n 3 -N 10 -c 1.0e-05

SAM_EXTRA_RG RG:TAGS
Users can add center, platform and other tags to their RG tag during bam creation. **It is strongly recommended that at least the platform is given, since some tools use that information.**

Default Value: (nothing)

MARK_DUPLICATE_PARAMS flags
Specify any additional parameters passed to picard during marking duplicates. By default, ASAP simply marks duplicates but doesn't remove them. Users should consult the picard documentation to determine which settings are appropriate for their needs.

Default Value: REMOVE_DUPLICATES=false ASSUME_SORTED=true VALIDATION_STRINGENCY=SILENT MAX_SEQUENCES_FOR_DISK_READ_ENDS_MAP=50000 OPTICAL_DUPLICATE_PIXEL_DISTANCE=100 CREATE_MD5_FILE=false

Realignment Settings

REALIGNMENT_TEMPLATE (--realignment-template) template-selection
Indicates which template to use during realignment. Currently, only GATK is provided.

Default Value: gatk

Realignment Settings (GATK)

GATK_REALIGNER_FLAGS flags
Specifies the parameters passed to GATK using RealignerTargetCreator. Users should consult the GATK documentation for details on which settings best suit their needs.

Default Value: --minReadsAtLocus 4 --windowSize 10 --mismatchFraction 0.15 --maxIntervalSize 500

GATK_INDEL_REALIGNER_FLAGS flags
Specifies any additional settings to be passed to GATK using IndelRealigner. Users should consult the GATK documentation for details on which settings best suit their needs.

Default Value: -compress 5 -LOD 5.0 --entropyThreshold 0.15 --maxConsensuses 30 --maxReadsForConsensuses 120 --maxReadsInMemory 350000 --maxReadsForRealignment 250000

REALIGN_CHROM_PARTITION chromosome-partition-string
Allows the user to define which chromosomes to be grouped during realignment. Realignments on grouped chromosomes will be done serially. Grouping avoids producing too many jobs.

Default Value: 1,2,3,4,5,6,7,8,9,10,11,12,13,14+15,16+17,18+19,20+21+22+MT+Y,X

Recalibration Settings

RECALIBRATION_TEMPLATE (--recalibration-template) template-selection
Indicates which template to use during recalibration. Currently, only GATK is provided.

Default Value: gatk

Recalibration Settings (GATK)

Please note that in some cases, ASAP's default settings include illumina specific settings.

RECAL_CHROMOSOME_PARTIONING chromosome-partition-string
Partition describing the chromosome groupings used during recalibration.

Default Value: 1,2,3,4,5,6,7,8,9,10,11,12,13+14+15,16+17+18,19+20+21+22+Y+MT,X

GATK_COUNT_COV_FLAGS flags
Specifies any additional settings to be passed to GATK using CountCovariatesWalker. Users should consult the GATK documentation on which settings are appropriate for their needs.

Default Value: -cov ReadGroupCovariate -cov QualityScoreCovariate -cov CycleCovariate -cov DinucCovariate --default_platform illumina

GATK_RECAL_FLAGS flags
Specifies any additional settings to be passed to GATK using TableRecalibrationWalker. Users should consult the GATK documentation on which settings are appropriate for their needs.

Default Value: --default_platform illumina --preserve_qscores_less_than 5 --smoothing 1

Move Bams Settings

The MOVE_BAMS step moves the highest level BAM files into the final directory. Once the files have been moved, symbolic links are made in case users need to refer back to the original BAM location for any reason. Users must keep in mind that if a link is broken, the file becomes invalid.

MOVE_TEMPLATE template-selection
Currently, only one template is provided, cp. This copies the links using unix cp, then deletes the original and replaces it with a symlink, once the copy has completed successfully.

Default Value: copy

MOVE_JOB_COUNT integer
This is effectively allows the user to control the maximum number of moves that will be performed simultaneously. if the move is performed across different file systems, too many independent move jobs

could slow one another down and result in longer execution times. If the move is within the same file system, it will be almost instantaneous and doesn't need more than a single job to complete quickly. It is recommended to use 1 or only a small number for this setting.

Default Value: 1

Variant Calling

When calling variants, users can choose different methods for calling SNPs and INDELs. These two are processed independently and the output resides in separate files upon completion.

SUBREGION_SIZE integer
Specifies the target size of subregion to use during variant calls. If the tail would be smaller than 50% the specified size, the chromosome is split evenly, resulting in slightly larger subregions.

Default Value: 25000000

SNP_CALLER (--snp-caller) template-selection
Allows the user to choose which template to use during variant calling. Current options include: gatk, samtools & glf_multiples

Default Value: gatk

INDEL_CALLER (--indel-caller) template-selection
Allows the user to choose which template to use during variant calling. Current options include: gatk & samtools

Default Value: gatk

Variant Calling (GATK)

GATK_SNP_FLAGS flags
Specifies any additional settings to be passed to GATK using UnifiedGenotyper to call SNPs. Users should consult the GATK documentation on which settings are appropriate for their needs.

Default Value: -l INFO -A DepthOfCoverage -A AlleleBalance -G Standard -mbq 20 -mmq 20 -stand_call_conf 50.0 -stand_emit_conf 20.0 -dcov 1000 -deletions 0.05 -gt_mode DISCOVERY -out_mode EMIT_VARIANTS_ONLY -nsl

GATK_INDEL_FLAGS flags
Specifies any additional settings to be passed to GATK using UnifiedGenotyper to call INDELs. Users should consult the GATK documentation on which settings are appropriate for their needs.

Default Value: -l INFO -A DepthOfCoverage -A AlleleBalance -G Standard -mbq 20 -mmq 20 -stand_call_conf 50.0 -stand_emit_conf 20.0 -dcov 1000 -deletions 0.05 -gt_mode DISCOVERY -out_mode EMIT_VARIANTS_ONLY -nsl

Variant Calling (Glf Multiples)

GLF_MULTIPLES_PARAMS flags
Specifies any additional settings to be passed to GLF Multiples during the Variant Calling.

Default Value: (nothing)

Variant Calling (Samtools)

ASAP uses a variation of the commands shown in the samtools documentation for calling variants. This command is truncated slightly, but yields the same final VCF (but does not keep the bcf.) Effectively, the call is in the form:

```
samtools mpileup (params) ref bam(s) | bcftools view (params) | \
vcfutils.pl varFilter (params) > new.vcf
```

MPILEUP_PARAMS flags
Specifies any additional settings to be passed to samtools/mpileup when producing Variant Calls. Users should consult the samtools documentation for settings appropriate for their needs.

Default Value: -uf

BCFTOOLS_PARAMS flags
Specifies any additional settings to be passed to bcftools when producing Variant Calls. Users should consult the samtools documentation for settings appropriate for their needs.

Default Value: -vcg

VCFUTILS_PARAMS flags
Specifies any additional settings to be passed to vcfutils.pl when producing Variant Calls. Users should consult the samtools documentation for settings appropriate for their needs.

Default Value: -D100

Move VCF

COMBINE_TEMPLATE (--combine-template) template-selection
Specify which template to be used. Currently, only one template is provided, GATK. Currently, this will concatenate chromosomes which were split using SUBREGION_SIZE smaller than the chromosome length. In future versions, it will also have the capability to perform merges for VCFs which were called in batches or by samples. This merge will use gatk, which it's fully implemented.

Default Value: gatk

VCF_COMBINE_MAX_JOBS (--vcf-combine-max-jobs) integer
Specifies maximum number of combine jobs that will be run at one time. This defaults to 1, and should probably never get much higher than 2 or 3 since the jobs themselves should take too long.

Default Value: 1

Annotation

ANNOTATE_VCF_TEMPLATE (--annotate-vcf-template) template-selection
Specify the template used to perform annotation. Currently, only one template is provided: *annovar.*

Default Value: annovar

Annovar Parameters

CONVERT_TO_ANNOVAR_PATH (--convert2annovar) file-path
Specifies the full path to the perl script, convert2annovar.pl which is part of the ANNOVAR suite. By default, ASAP assumes that the program is in the user's PATH.

Default Value: convert2annovar.pl

ANNOTATE_VARIATION_PATH (--annotate-variation) file-path
Specifies the full path to the perl script, annotate_variation.pl which is part of the ANNOVAR suite. By default, ASAP assumes that the program is in the user's PATH.

Default Value: annotate_variation.pl

ANNOVAR_BUILD_VER string
The UCSC build number associated with the user's reference sequence.

Default Value: hg19

ANNOVAR_GENE_OPTIONS string
Options passed to annotate_variation.pl for gene annotation.

Default Value: (nothing)

ANNOVAR_REGION_OPTIONS complex-string
Options to be passed to annotate_variation.pl during region annotations. The format for this variable is tricky. Entries are separated by a comma, ",". Each entry consists of two pieces of information separated by a colon: region_database:flag. Please see ANNOVAR_FILTER_OPTIONS for an example

Default Value: (nothing)

ANNOVAR_FILTER_OPTIONS complex-string
Options to be passed to annotate_variation.pl during ANNOVAR filtering.

For example: *avsift:-sift 0,ljb_all:-otherinfo* represents 2 distinct entries. One for the database avsift and one for ljb_all. When performing the annnotation for avsift, the options, *-sift 0*, flag is passed. When using the database ljb_all, *-otherinfo* is passed.

Default Value: avsift:-sift 0,ljb_all:-otherinfo

ANNOVAR_GENE_DBS (--annovar-gene-db) comma-separated-list
Comma separated list of gene databases to be used. These should be spelled the same as would be used when following -dbtype.

Default Value: gene

ANNOVAR_REGION_DBS (--annovar-region-dbs) comma-separated-list
Comma separated list of region databases to be used. These should be spelled the same as would be used when following -dbtype.

Default Value: avsift,ljb_all,ljb_pp2,ljb_lrt,ljb_mt,ljb_phylop,snp135,1000g2010nov_all,esp5400_all

ANNOVAR_FILTER_DBS (--annovar-filter-dbs) comma-separated-list
Comma separated list of regino databases to be used. These should be spelled the same as would be used when following -dbtype.

ANNOVAR_DB_PATH (--annovar-db-path) path
Full or relative path to the directory containing the ANNOVAR databases. This can be shared by multiple ASAP projects, and should not interfere with being used by non-ASAP based scripts. It should be noted, that ASAP does write a few files to this directory that are used to keep up with how recent the databases are.

Default Value: (nothing)

ANNOVAR_WEB_DBUPDATES comma-separated-list
This is a list of ANNOVAR databases which require the -webfrom flag. Users might have to tweak these, depending on the changes to ANNOVAR's database support and the age of the ASAP installation.

Default Value: ljb_all,ljb_pp2,ljb_lrt,ljb_mt,ljb_phylop,esp5400_all

ANNOVAR_UPDATE_FREQ integer
Specifies the maximum number of days between downloading the databases selected before downloading again. ASAP will download any new databases it encounters, and make a note of the date it was updated. After N days, if the user generates an annotation script, it will download the update once again.

As a result of the database download, this can take quite a few minutes. Users should be aware of this when running on gateway machines with restricted time limits.

Default Value: 30

ALTERNATE_DB_NAME complex-string
This is effectively defines a lookup table to help ASAP translate ANNOVAR calls where the database download name doesn't match the name of the dbtype specified during annotation. As a result, it's a little tricky. Basically, entries are separated by semicolons, ";". Each entry consists of 2 members separated by a single colon, ":". The right hand side of the entry can contain a list, which is to be separated by ",". For example, "1000g2010nov_all:1000g2010nov;1000g2012feb_all:1000g2012feb" translates to 2 entries. 1000g2010nov_all and 1000g2012feb_all are used by -dbtype, and their counterparts, 1000g2010nov and 1000g2012feb are used by the download command. If there were multiple downloads required for one of the dbtypes, the second portion would have contained 2 or more entries separated by a comma.

Default Value: 1000g2010nov_all:1000g2010nov;1000g2012feb_all:1000g2012feb

Job Reporting

--lines (-L) integer
Used only for SHOW_LOG report, users can specify how many lines are shown from the ends of the error and out log files.

Default Value: 10

Importing Bams (IMPORT_BAM_FILES)

As of version 1.1.3, users can choose between finding bams using regular expressions and explicitly describing the chromosome, sample ID as they can be found in the BAMs.

--import-as step
When importing data, users can specify which step the data will appear to be. By default, the step is AlignmentSplit, however, users can import the data as realigned or recalibrated, depending on their needs for the imported data, as well as the processing required for other data in the project.

Options include: AlignmentSplit, RealignBatch, RealignSample and RecalibrateSample. It is imperative that the specified step be one of the steps used during processing, otherwise, the imported data will be ignored during script generation.

Default Value: AlignmentStrip

Importing Bams - Regular Expression Search

--bam-directory path
(required) This directs ASAP to the directory inside which the bam files of interest can be found. Bam files can be nested in subdirectories.

--bam-format regex
Regular expression describing the format of the filenames. The expression must capture at least the sample ID and can optionally capture chromosome as well, if the file contains only one chromosome. ASAP accepts only bam files that contain 1 sample per bam. Each sample can be contained within a single file (with all chromosomes in that file) or in in a separate file for each chromosome.

Default Value: ([A-Za-z0-9]+)_chr([0-9]+|X|Y|MT).bam

--bam-field-order comma-separated-list
Indicates the interpretation behind the captured text in the --bam-format expression. Options include: SAMPLE and CHR. SAMPLE is required, and the values must be in the same order as they are found in the expression.

Default Value: SAMPLE,CHR

Importing Bams - Flat File

--bam-list filename
Filename contains 3 fields, chromosome(s), sample ID and bam filename, all separated by tabs. If there is more than 1 chromosome for a given sample in the file, each chromosome should appear in the same column, separated by '+' signs. For example, 1+2+3+4+5.

If the user specifies a bam-list, the regular expression settings are ignored during import.

Default Value: (nothing)

10. Common Errors

Reference Sequence .dict is missing

This file must be created according to the instructions in System Preparation.

bwa fails during alignment due to missing .bwt file

This file is required by bwa during alignment and must be produced according to the instructions in System Preparation.

Realignment fails "lock count not be obtained"

```
* ##### ERROR MESSAGE: Couldn't read file /home/###/reference/human_g1k_v37.fasta because
Could not open index file because a lock could not be obtained.
```

If you see something like the above in your error logs, then multiple copies of GATK are attempting to create the .fai file for the reference. This can be fixed by following the instructions in System Preparation.

11. Extending ASAP

Asap has been developed in a way that requires no modifications to the pre-existing code for certain types of new functionality. Users can use this feature to add or make subtle variations of the application to suit their own needs in three different ways: New Configuration Objects, New Steps and New Step Templates.

New Configuration Objects

Inside the directory, src/modules/ngs/config are a few ruby scripts. Each of these define a grouping of application-wide configuration objects. In general, users won't need to add new configuration objects, unless they are creating steps which share configuration parameters. An example of a shared configuration parameter is the reference sequence.

To create a new configuration object, the user must follow the following guidelines:

The user should create a new class using standard ruby CamelCase convention. This class must be implemented in a file within the directory, src/modules/ngs/config, under the same name as the class, except all letters be lower case, and words separated by a single underscore, "_", character. For example, the file, src/modules/ngs/config/gatk_parameters.rb defines a class GatkParameters, which is responsible for managing the variables associated with GATK.

```
class ExternalApplications
    include Configurable

    attr_accessor :picard_path

    # Manage all of the configuration parameters
    # associated with bwa, samtools and other external applications
    def initialize_configuration
        cfg_class = "REQUIRED_APP_PATHS"
        register_config_class(cfg_class, "These paths must be set in order for execution to be
continue.")
        register_variable("", "picard-path", "PICARD_PATH", cfg_class, "The path to each of the
picard tools (MarkDuplicates, FilterSamReads, etc)", "picard_path", "")
    end
end
```

This new class should use the mixin, Configurable. This grants the new class access to the Configurable functions necessary to registering it's parameters. Below is a snippet from the ExternalApplications class, which is one of the configuration objects delivered with ASAP (I've removed many lines to make it less cluttered). .

Each file within the config directory will be "discovered" when the application launches. ASAP will then initialize the object, whose class-name is inferred based upon the filename and then the function, initialize_configuration is called. This allows the new configuration object to register it's variables.

The following functions are available for all configurable objects:

Function Name	Parameter List	Purpose
register_config_class	**name** - This is just the name associated with this "class" of configuration variables **description** - This is the content written to sample configuration file as the "header" for the group of variables.	This sets up a grouping for the configuration class. This grouping will then be used when generating sample configurations. The name is used when assigning new variables to the configuration class and the description is what is shown as header comments within the sample configuration.
register_variable	**sflag** - short cmdline flag **lflag** - long cmdline flag **name** - variable name (inside configuration file) **cfg_class** - configuration class this variable is assigned to **comment** - comment which will be written to the sample configuration describing this variable **variable** - the variable name to be assigned within this object **default_value** - the default value. If left blank, the variable will be commented out by default and be nil unless explicitly set. **conversion** - function that can be used to convert the variable to it's proper type (by default, all items in the configuration file are strings)	This function will add a new variable to the configuration manager. The configuration manager will use the *flags* (if not empty strings) to allow users to set the variable via command line arguments, or the *name* property to pull contents from the configuration file and environment. Any of those can be an empty string, "", but at least one must contain a meaningful string. It is up to the user to ensure that no flags are redundant with other potentially active configurables. The default value should be a string value (or nil, which is the default). This is the value that the variable will have until it is set either by configuration or command line arguments. Conversion should be a simple function call that can be made on a string to return the desired type. An example might be to_i, which would convert the contents to an integer.

The user's new configuration object is instantiated under a global variable named $UNDERSCORE_NAME, which is the class name in underscore notation prefixed by a $. So, once the file described above were loaded, the following object would exist: $EXTERNAL_APPLICATIONS. If the user wants to expose those variables, they should provide appropriate accessor/mutators, which can be done using rubies attr_accessor method. In the class above, there are one variable exposed: $EXTERNAL_APPLICATIONS.picard_path

Calling this function will return the user's selection (or the default value) for variable PICARD_PATH.

Adding New Templates

Templating allows ASAP to use the same basic step, but apply it in different ways. For example, the user can choose to call variants using GATK, samtools as well as glf_multiples. The basics between all three are pretty much the same: We have a bunch of bam-files, from which we want to extract variants, but the way we use each program is going to be quite different.

Just like Configuration and Steps, ASAP uses self discovery for templates as well. As such, users can copy a pre-existing template, make changes to it, and as long as the new file is named according to ASAP's requirements, the new template will be automatically available the next time the user runs.

In a later version of this document, we'll define all of the variables and functions which users can take advantage of in order to create their own templates from scratch. However, we can quickly give users the ability to reuse existing templates for their own needs with just a few basic instructions.

Naming Conventions for Templates

There are 2 parts of the template discovery process. First, the step object (such as "RealignBatch") will probe for templates inside the directory src/modules/ngs/steps/templates/, which are named in the following way: realign_batch.????.erb. Those question marks represent the name of the template as we'll expect to see it from the configuration. By default, ASAP comes with one template for batch realignment: realign_batch.gatk.erb.

In some cases, a step might be able to reuse another step's template. ASAP uses this approach for calling variants. Users can call variants by sample and by batch as well as calling variants across all samples in the current set of data. However, the actual steps required to perform those tasks is identical. In those cases, CallVariantsByBatch and CallVariantsBySample both use the CallVariants templates, unless the object first finds templates which match the default case.

Embedded Ruby (ERB) and variable substitution

ERB allows developers to embed ruby statements into a template using a few sets of delimiters:

- % ruby code
- <% ruby code %>
- <%= ruby code that returns value %>

The first form is probably the least common. In this case, a single % sign at the front of the line indicates that the remaining text is to be treated as ruby. The other two enclose a ruby statement inside the opening and closing delimiters. The last two differ only by the = sign in the latter, which indicates that we want to capture the output of the ruby statement and write it to the template. This is often used set up filenames and paths that will be used by the scripts during their execution. The other just executes ruby code. It might return a value, but that value will not be embedded into the resulting script.

Aside from these special tags, the rest of the template is just regular ruby code. The executor will add a few functions which will be specific to the execution type, but otherwise, the contents of the scripts produced by asap will come from these template files.

Example Template Modification

We are currently working to add INDEL extraction as a step that will run parallel to SNPs, allowing users to extract SNPs and INDELs into separate files. However, if a user currently wants to extract INDELs or both SNPs and INDELs at the same time using GATK, it's actually quite an easy accomplishment.

This example is a bit outdated. We will provide a more complete example if sufficient interest arises. Users can still get a general idea of how to perform this modification and look at the differences between some of the templates for SNP or INDEL calling.

First, we will copy the file, src/modules/ngs/steps/templates/call_snps.gatk.erb to a new name, call_snps.gatk_indels.erb, into the same directory as the original. Once that is done, you have one more change to make. Open the new file up in your favorite editor and move down to line 23, which should look like this:

```
`<%= gatk_vcf_calls %> -glm SNP -L #{$chr}:#{range} -I #{$bam_files.join(" -I ")} -metrics #
{metric_filename} -o #{output_filename}`
```

(except it should all be on one line and have no special color, unless your editor knows erb or ruby syntax.) I've highlighted the one flag you need to change using a yellow highlight. That line is the one doing all of the work. It will pull the GATK settings for VCF calls, identifies which chromosome and the range to be used and provides GATK with all of the bam files that this script needs to call the SNPs like you wanted. However, we want to change that to do INDELs. So, we make one change. Those part I've highlighted above is the flag used to tell GATK to produce SNPs. To produce INDELS, we would need to change that to -glm INDEL. If you were to want to change that extract SNPs and INDELs into the same file, you can change it to -glm BOTH.

Once you've made your change, just save it, and in your configuration, assign gatk_indels to the configuration setting, VARIANT_CALLER.

ASAP doesn't need any other information to know about the new template. It will look for files named step_name.template_name.erb and decide which templates each step have available. So, once you overcome the strange syntax in those template files, you can very quickly add new variations to your pipeline without really very much code at all.

In a future version of this document, we'll provide clear details about the syntax used in these templates (it's called eRuby, or ERB), documentation on all of the variables exposed at each step, as well as how to create entirely new steps.

12. Installing Ruby into $HOME

Users can install ruby using RVM, ruby version manager. RVM was designed to be installed in the user's home directory, which means the user doesn't have to have root privileges to install it, nor additional versions of ruby. Below is a walk-through demonstrating how to install ruby using RVM.

First, download and install RVM:

```
$ curl -L get.rvm.io | bash -s stable
```

Conveniently, this is all it takes to download and install rvm into your home directory. Follow the directions listed toward the bottom which instruct you to source the appropriate file necessary to *activate* rvm.

```
$ source ~/.rvm/scripts/rvm
$ which rvm
~/.rvm/bin/rvm
```

Now that we know it is installed, we need to install a recent version of ruby. As of the time I'm writing this document, 1.9.3 is the latest stable release:

```
$ rvm install 1.9.3
Fetching yaml-0.1.4.tar.gz to /home/vandy/.rvm/archives
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  460k  100  460k    0     0   284k      0  0:00:01  0:00:01 --:--:--  452k
                                                ...
```

Finally, the last bit of installation is the sqlite3 gem. The directions below assume that the developer packages for sqlite3 are currently installed on the system you are using. If that isn't true, then you will need to consult the gem install documentation or ask your system administrator for assistance (in general, this is an easy task for system admins these days).

```
$ gem install sqlite3
Fetching: sqlite3-1.3.6.gem (100%)
Building native extensions.  This could take a while...
Successfully installed sqlite3-1.3.6
1 gem installed
Installing ri documentation for sqlite3-1.3.6...
Installing RDoc documentation for sqlite3-1.3.6...
```

If all of that went smoothly, then you can run a really quick test to ensure that everything is working as it should:

```
$ ruby -e "warn require 'sqlite3'"
true
$ which ruby
~/.rvm/rubies/ruby-1.9.3-p194/bin/ruby
```

If that first command prints “true” like you see above, then the version of ruby you just installed works and the sqlite3 gem is properly installed.

That last statement simply returns the path to this successfully installed version. You will want to take note of that return value and use it for the RUBY_PATH setting once you generate your ASAP configuration.