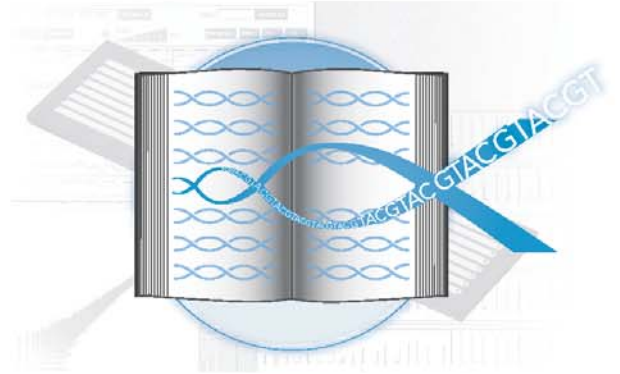


# Genome Analyzer Pipeline Software Version 1.0 User Guide

FOR RESEARCH ONLY







# Notice

This publication and its contents are proprietary to Illumina, Inc., and are intended solely for the contractual use of its customers and for no other purpose than to operate the system described herein. This publication and its contents shall not be used or distributed for any other purpose and/or otherwise communicated, disclosed, or reproduced in any way whatsoever without the prior written consent of Illumina, Inc.

For the proper operation of this system and/or all parts thereof, the instructions in this guide must be strictly and explicitly followed by experienced personnel. All of the contents of this guide must be fully read and understood prior to operating the system or any of the parts thereof.

**FAILURE TO COMPLETELY READ AND FULLY UNDERSTAND AND FOLLOW ALL OF THE CONTENTS OF THIS GUIDE PRIOR TO OPERATING THIS SYSTEM, OR PARTS THEREOF, MAY RESULT IN DAMAGE TO THE EQUIPMENT, OR PARTS THEREOF, AND INJURY TO ANY PERSONS OPERATING THE SAME.**

Illumina, Inc. does not assume any liability arising out of the application or use of any products, component parts, or software described herein. Illumina, Inc. further does not convey any license under its patent, trademark, copyright, or common-law rights nor the similar rights of others. Illumina, Inc. further reserves the right to make any changes in any processes, products, or parts thereof, described herein without notice. While every effort has been made to make this guide as complete and accurate as possible as of the publication date, no warranty or fitness is implied, nor does Illumina accept any liability for damages resulting from the information contained in this guide.

© 2008 Illumina, Inc. All rights reserved. **Illumina, Solexa, Making Sense Out of Life, Oligator, Sentrix, GoldenGate, DASL, BeadArray, Array of Arrays, Infinium, BeadXpress, VeraCode, IntelliHyb, iSelect, CPro, iScan, and GenomeStudio** are registered trademarks or trademarks of Illumina. All other brands and names contained herein are the property of their respective owners.





# Revision History

Part Number	Revision Letter	Date
1004759	A	June 2008
1003881	A	January 2008



# Table of Contents

Notice	iii
Revision History	v
Table of Contents	vii
List of Figures	xi
List of Tables	xiii
<b>Chapter 1 Overview</b>	<b>1</b>
Introduction	2
Additional Information	2
Genome Analyzer Pipeline Software Workflow	3
Installation	3
Running the Analysis	3
Analysis Output	3
Reporting Problems	4
Technical Assistance	4
<b>Chapter 2 Core Concepts</b>	<b>5</b>
Introduction	6
Analysis Modules	6
Understanding the Run Folder	8
Run Folder Structure	9
Images Folder	10
Data Folder	10
Run Folder Naming	11
File Naming	12
Parameters	12
Paired Reads	12
Calibration and Input Parameters	13
Image Offsets	13
Frequency Cross-Talk Matrix	14
Phasing/Prephasing Estimates	15
Sample Information	15
Alignment Algorithms	16

<b>Chapter 3</b>	<b>Running the Analysis</b>	<b>17</b>
	Introduction	18
	Starting the Genome Analyzer Pipeline Software	18
	Running a Standard Analysis	19
	Specifying the IPAR Folder	19
	Parallelization Switch	20
	Nohup Command	20
	Command Line Options	21
	General Options	21
	GOAT Options	22
	GOAT and Bustard Options	22
	Paired Reads	23
	IPAR Analysis	23
	Makefile Targets	24
<b>Chapter 4</b>	<b>Using GERALD</b>	<b>27</b>
	Introduction	28
	GERALD Parameters	29
	ANALYSIS Variables	29
	Analysis Parameters	30
	Filtering Parameters	31
	USE_BASES Option	31
	Lane-by-Lane Parameters	32
	FORCE Option	33
	Rerunning the Analysis	33
	Contaminant Filtering	33
	Building an SRF Archive	33
	GERALD Configuration File	35
	Lane-Specific Options	36
	Optional Parameters	36
	Paired-End Analysis Options	37
	Preparing the Reference Genome	38
	ELAND Alignments	40
	Missing Bases in ELAND	41
	Using ANALYSIS eland_tag	41
	Using ANALYSIS eland_extended	42
	Using ANALYSIS eland_pair	43
<b>Chapter 5</b>	<b>Analysis Output</b>	<b>47</b>
	Introduction	48
	Visual Analysis Summary	48
	Results Summary	48
	Cluster Intensity	56
	Error Rates	57
	Text-Based Analysis Results	58
	Interpretation of Run Quality	60
	Summary.htm	60
	IVC.htm	64
	All.htm and Error.htm	64



<b>Chapter 6</b>	<b>Advanced Pipeline Usage</b>	<b>65</b>
	Introduction	66
	Running Bustard as a Standalone Program	66
	Assigning a Control Lane	66
	Running GERALD as a Standalone Program	67
	Additional "Make" Options	67
	Running ELAND as a Standalone Program	68
	Compiling ELAND	69
	Command Line Syntax	69
<b>Appendix A</b>	<b>System Requirements and Software Installation</b>	<b>71</b>
	Introduction	72
	System Requirements	72
	Network Infrastructure	72
	Analysis Computer	73
	Installation Prerequisites	75
	Setting Up Email Reporting	75
	Installing the Pipeline Software	77
	Compiling on Other Platforms	77
	Directory Setup	77
<b>Appendix B</b>	<b>Output File Descriptions</b>	<b>79</b>
	Introduction	80
	Output File Types	80
	Intensity Files	81
	Sequence Files	81
	Quality Score Files	82
	Efficiency	82
	Intermediate Output Data Files	83
	Output File Formats	86
	Parameters File Format	89
<b>Appendix C</b>	<b>Using Parallelization</b>	<b>93</b>
	Introduction	94
	"Make" Utilities	94
	Standard "Make"	94
	Distributed "Make"	94
	Customizing Parallelization	94
	Parallelization Limitations	97
	Memory Limitations	98
<b>Appendix D</b>	<b>Base Call Calibration and Alignment Scoring</b>	<b>99</b>
	Introduction	100
	Goal	100
	Method	100
	Modifications to the Phred Formula	100
	Characteristics of the Quality Scores Produced by the Base Caller	101
	High Quality Scores	101

- Limitations of the Recalibration Method . . . . . 101
  - Major Alignment Errors . . . . . 101
  - SNP Rate . . . . . 101
  - Size of Data Set . . . . . 101
- General Usage . . . . . 102
  - Configuring Quality Table Sources in GERALD. . . . . 102
- Expert Usage . . . . . 104
  - Extracting Quality Predictors. . . . . 104
  - Extracting Reference Bases . . . . . 106
  - Creating a Quality Table . . . . . 107
  - Generating New Quality Values . . . . . 109
  - Configuring Quality Table Sources in GERALD. . . . . 111
  - Default and Experimental Predictors. . . . . 112
  - Further Considerations . . . . . 113
- Frequently Asked Questions . . . . . 114



# List of Figures


Figure 1	Three Steps of Data Analysis . . . . .	2
Figure 2	Pipeline Modules . . . . .	6
Figure 3	Run Folder Directory Structure . . . . .	8
Figure 4	Frequency Cross-Talk Matrix and Phasing File Locations . . . . .	14
Figure 5	Run Folder Structure and Output File Types . . . . .	80
Figure 6	Q vs. Qphred . . . . .	114
Figure 7	Actual vs. Computed Error Rate for Three Sets of Simulated Reads . . . . .	116



# List of Tables

Table 1	Illumina Technical Support Contacts . . . . .	4
Table 2	File Naming Components . . . . .	12
Table 3	ANALYSIS Variables . . . . .	29
Table 4	Analysis Parameters . . . . .	30
Table 5	USE_BASES Options . . . . .	32
Table 6	Lane-by-Lane Parameters. . . . .	32
Table 7	GERALD Configuration File Parameters . . . . .	35
Table 8	GERALD Configuration File Lane-Specific Options . . . . .	36
Table 9	GERALD Configuration File Optional Parameters . . . . .	36
Table 10	GERALD Configuration File Paired-End Analysis Options. . . . .	37
Table 11	Parameters for ANALYSIS eland_extended . . . . .	43
Table 12	Parameters for ANALYSIS eland_pair . . . . .	44
Table 13	Example of Individual Alignments Table . . . . .	52
Table 14	Example of Unique Paired Alignments Table . . . . .	53
Table 15	Example of Unique Paired Alignment Effects Table . . . . .	54
Table 16	Example of Non-unique Paired Alignments Table. . . . .	54
Table 17	Example of Mismatching Rate Table . . . . .	54
Table 18	Example of Relative Orientation Statistics Table . . . . .	55
Table 19	Example of Insert Size Statistics Table. . . . .	55
Table 20	Example of Insert Statistics Table . . . . .	56
Table 21	Text-Based Analysis Results . . . . .	58
Table 22	Example of Lane Results Summary . . . . .	60
Table 23	Example of Expanded Lane Summary . . . . .	60
Table 24	Options for ELAND_standalone.pl. . . . .	68
Table 25	Data Volumes Per Experiment. . . . .	72
Table 26	Intermediate Output File Descriptions . . . . .	83
Table 27	Contaminant Filtering-Specific Files . . . . .	85
Table 28	Final Output File Formats . . . . .	86
Table 29	Intermediate Output File Formats . . . . .	87
Table 30	QCAL_SOURCE Variable Values . . . . .	102
Table 31	Default Base-Call Quality Predictors . . . . .	112
Table 32	Experimental Base-Call Quality Predictors. . . . .	113





# Chapter 1

## Overview

### Topics

- 2 Introduction
- 3 Genome Analyzer Pipeline Software Workflow
  - 3 Installation
  - 3 Running the Analysis
  - 3 Analysis Output
- 4 Reporting Problems
- 4 Technical Assistance

# Introduction

The Genome Analyzer Pipeline Software (Pipeline) is a set of utilities designed to perform a complete offline data analysis of a sequencing run. It is supplied as source code and scripts.

Data analysis consists of three steps: image analysis, base calling, and sequence analysis.

1. **Image analysis**—Uses the raw TIF files to locate clusters on the image, and outputs the cluster intensity, X,Y positions, and an estimate of the noise for each cluster. The output from image analysis provides the input for base calling.
2. **Base calling**—Uses cluster intensities and noise estimate to output the sequence of bases read from each cluster, along with a confidence level for each base.
3. **Sequence analysis**—Allows for alignment to a reference sequence, filtering of data based on predefined criteria, and visualization of the result.

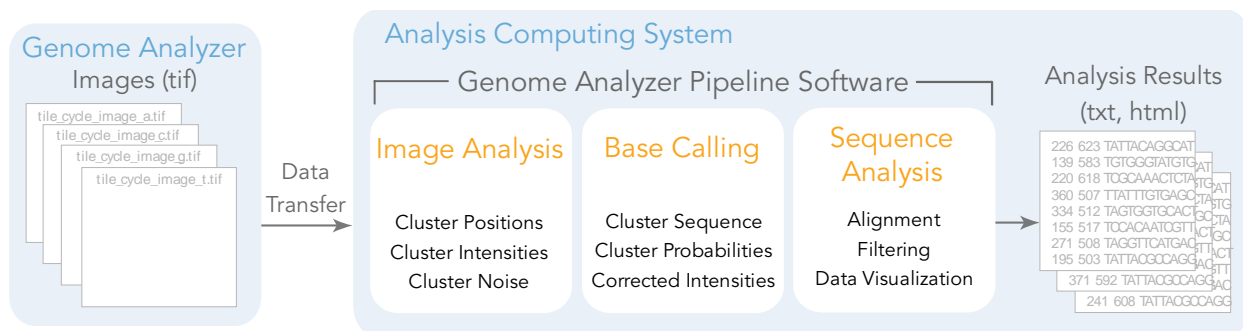


Figure 1 Three Steps of Data Analysis

The output data produced by the Genome Analyzer Pipeline Software are stored in flat text, tab-delimited files in a hierarchical folder structure called the Run Folder. The Run Folder includes all data folders generated from the Genome Analyzer and the data analysis structure. For a detailed description of the Run Folder structure, see *Understanding the Run Folder* on page 8.

The Pipeline requires a Linux system with specific processing and data storage capacity. For specific requirements, see *System Requirements* on page 72.

## Additional Information

Additional information on the Genome Analyzer Pipeline Software can be found in the Pipeline/docs folder of your Pipeline software distribution.



# Genome Analyzer Pipeline Software Workflow

The image data from a sequencing run are saved on the Genome Analyzer computer in a folder structure organized by cycle, lane, and tile number. The data are transferred to a network location for analysis after the sequencing run is complete or by mirroring the data to the storage location while the run progresses.

The following is an overview of the Pipeline workflow.

## Installation

1. Install the Pipeline prerequisites on a suitable Linux system. See *Installation Prerequisites* on page 75.
2. Install the Pipeline software and compile the Pipeline using the “make” command. See *Installing the Pipeline Software* on page 77.
3. Set up the “Instruments” directory for parameters files. See *Directory Setup* on page 77.

## Running the Analysis

1. Change to the Run Folder location.
2. Create a configuration file that specifies what analysis should be done for each lane. See *GERALD Parameters* on page 29 and *GERALD Configuration File* on page 35.
3. Run a check on the Run Folder. See *Running a Standard Analysis* on page 19.
4. Add command line options, generate the analysis folder, and corresponding makefiles. See *Command Line Options* on page 21.
5. Change to the analysis directory and start your analysis.

## Analysis Output

1. View the analysis results of your run. See *Visual Analysis Summary* on page 48 and *Text-Based Analysis Results* on page 58.
2. Interpret the run quality. See *Interpretation of Run Quality* on page 60.

## Reporting Problems

Contact Illumina Technical Support to report any issues with the Pipeline.

When reporting an issue, it helps to capture all the output and error messages produced by a run. This is done by redirecting the output using “nohup” or the facilities of a cluster management system. For an explanation of “nohup,” see *Running a Standard Analysis* on page 19.


It helps to attach the makefile corresponding to the part of the Pipeline that is causing the problem. If there are GERALD-related issues, it helps to post the config.txt file found in the GERALD output folder. For problems relating to specific tiles or files, it is useful to send the output of “wc -l” and “ls -l” on these files.

## Technical Assistance

For technical assistance, contact Illumina Technical Support.

**Table 1** *Illumina Technical Support Contacts*

Contact	Number
Toll-free Customer Hotline (North America)	1-800-809-ILMN (1-800-809-4566)
International Customer Hotline	1-858-202-ILMN (1-858-202-4566)
Illumina Website	<a href="http://www.illumina.com">www.illumina.com</a>
Email	<a href="mailto:techsupport@illumina.com">techsupport@illumina.com</a>



## Chapter 2

# Core Concepts

### Topics

- 6 Introduction
- 6 Analysis Modules
- 8 Understanding the Run Folder
  - 9 Run Folder Structure
  - 11 Run Folder Naming
  - 12 File Naming
  - 12 Parameters
  - 12 Paired Reads
- 13 Calibration and Input Parameters
  - 13 Image Offsets
  - 14 Frequency Cross-Talk Matrix
  - 15 Phasing/Prephasing Estimates
  - 15 Sample Information
- 16 Alignment Algorithms

## Introduction

Analysis modules perform the specific tasks of image analysis, base calling, and sequence alignment. During an analysis run, a defined folder structure is generated that captures the output of an instrument run in text files and parameters files. Parameters files contain calibration and input settings that optimize your analysis run and the alignment programs perform sequence analysis. This section describes these core concepts of the Genome Analyzer Pipeline Software.

## Analysis Modules

The Pipeline is divided into modules that are managed by the “make” utility. The “make” utility is commonly used to build executables from source code and is designed to model dependency trees by specifying dependency rules for files. These dependencies are stored in a file called a makefile. “Make” has a dual purpose within the Pipeline software:

- ▶ To build executables from source code
- ▶ To perform data analysis steps using the software

Each Pipeline module is a collection of Perl or Python scripts and C++ executables, and has its own makefile associated with the analysis task. The script `goat_pipeline.py`, named after the General Oligo Analysis Tool (GOAT) calls the subscripts for three Pipeline modules: Firecrest, Bustard (`bustard.py`), and GERALD (`GERALD.pl`).

Any of the first two scripts can invoke the next script automatically, so there is no need to call more than one script for any given analysis run. Typically, the analysis begins with the image analysis script, `goat_pipeline.py`. However, if you need to reanalyze data, you can start with one of the other scripts and use different parameters.

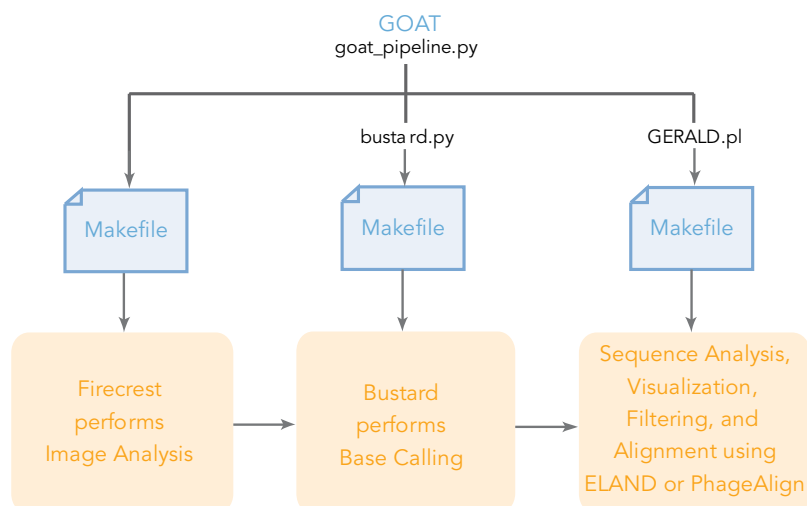


Figure 2 Pipeline Modules

- ▶ **Firecrest** is the module used for image analysis. Firecrest identifies cluster positions and extracts intensities. Through image filtering, it sharpens and enhances clusters, removes background noise, and detects clusters based on morphological features on the image. Firecrest also adjusts the scale and registration of an image.
- ▶ **Bustard** is the module used for base calling. Bustard deconvolves the signal from the clusters and applies correction for cross-talk, phasing, and prephasing.
  - **Frequency cross-talk**—The Genome Analyzer uses two lasers and four filters to detect four dyes attached to the four types of nucleotide, respectively. The frequency emission of these four dyes overlaps so that the four images are not independent. The frequency cross-talk is deconvolved using a frequency cross-talk matrix.
  - **Phasing/Prephasing**—Depending on the efficiency of the fluidics and the sequencing reactions, a small number of molecules in each cluster may run ahead (prephasing) or fall behind (phasing) of the current incorporation cycle. This effect is mitigated by applying corrections during the base calling step.
- ▶ **Generation of Recursive Analyses Linked by Dependency (GERALD)** is the module used for sequence alignment, data visualization, filtering, and alignment. The following two alignment programs work within the GERALD module:
  - **Efficient Large-Scale Alignment of Nucleotide Databases (ELAND)** is very fast and aligns for up to two errors from a reference for the first 32 bases. This algorithm is used for any reference larger than 100 Kbases.
  - **PhageAlign** does an exhaustive alignment (all possible alignments up to arbitrary edit distances), but is slow.

A run of the Pipeline is a two-stage process:

1. Generate the folders and makefiles using one of the above scripts.
2. Start the Pipeline analysis by executing “make.” See *Running a Standard Analysis* on page 19 for details.

In addition, the rest of this section describes the analysis and input parameters, and the command line options used in an analysis run.

## Understanding the Run Folder

The Pipeline operates in a specific directory called the Run Folder where the images and analysis output files are saved by default in a hierarchical structure.

The following figure illustrates a typical Run Folder.

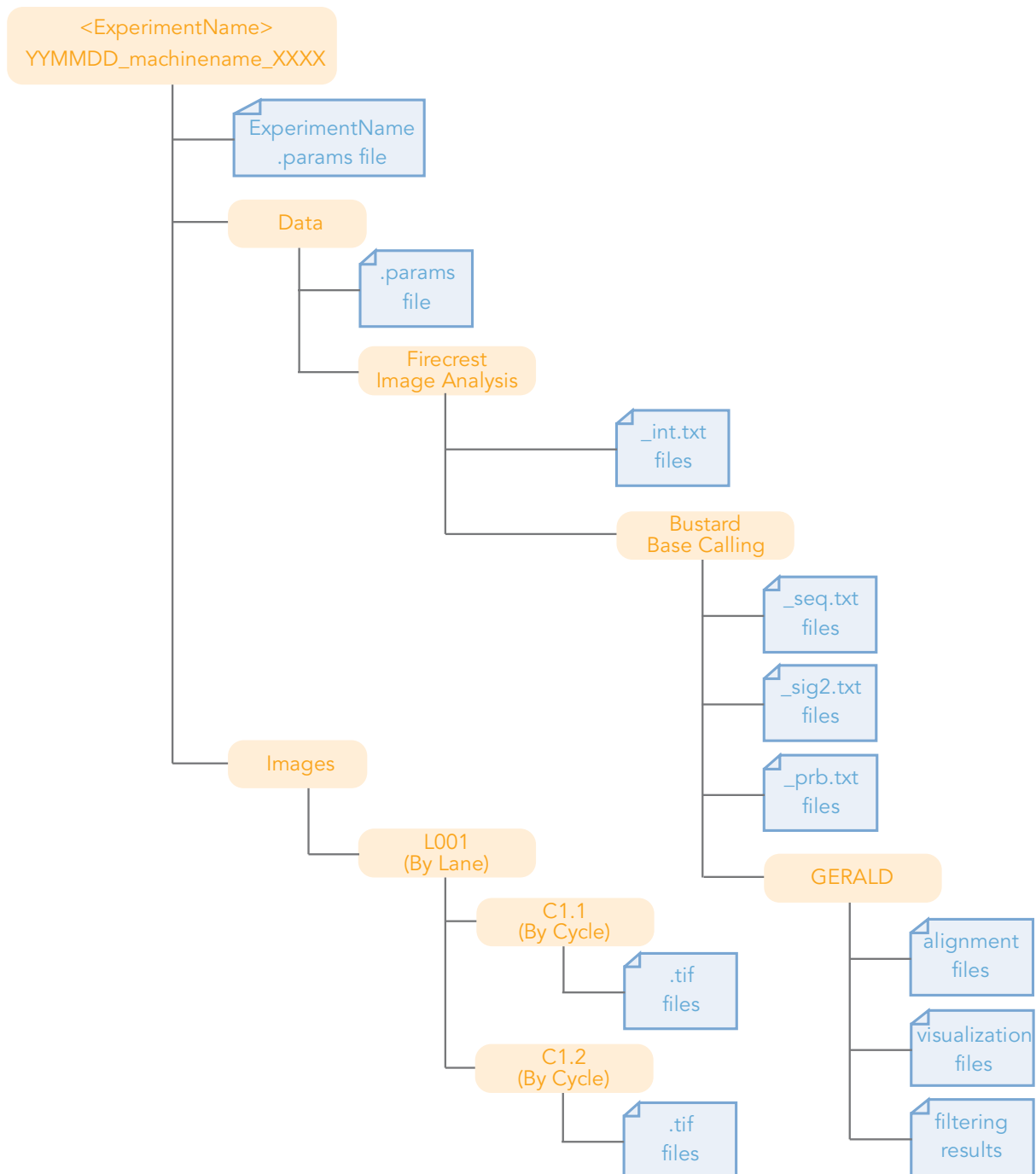


Figure 3 Run Folder Directory Structure

The standardized structure, file naming conventions, and file formats of the Run Folder allow for the following:

- ▶ A single point of data storage, logging, and analysis output during and after a run.
- ▶ Encoding sufficient information to trace the history of the data in the Run Folder back to the laboratory notebook without confusion between instruments, experiments, or sites.
- ▶ Standardized input and output enabling component software to operate without error, regardless of the instrument generating the data.
- ▶ Capturing and encoding enough information to independently reanalyze the data at any time, in such a way that existing extractions of sequence and related data are preserved, and parameters used during any point of the extraction process are captured and related to the subsequent output data.
- ▶ Subsequent analyses to be stored in the Run Folder.
- ▶ The software tools and other user software to implement and enforce these structures and standards.

## Run Folder Structure

The Run Folder contains the Images folder and Data folder as illustrated in Figure 3. The Data folder contains Image Analysis folders and the Image Analysis folders contain Sequence folders.

- ▶ The Images folder holds the images from every tile for a given cycle of sequencing.
- ▶ The Data folder is created the first time analysis is initiated for a given experiment. Any analysis performed on the data is saved within the Data folder.

Each run of the main analysis modules creates a subdirectory in the Data folder of the Run Folder as follows:

- ▶ Each run of the image analysis software (Firecrest) creates a new image analysis output folder in the Data folder.
- ▶ Each run of the base calling software (Bustard) creates a new subdirectory in the image analysis subdirectory on which the base calls are based, resulting in a tree-like structure of analyses.
- ▶ Parameters and versions for any given analysis run are logged in the folder structure to make it possible to reconstruct any previous analysis run.

You can do multiple analyses of the data using different analysis parameters and the results will not be overwritten. The default naming convention consists of the number of cycles run, the version of the software used for the operation (Firecrest, Bustard), the date the analysis initiated, and the login of the user. If the user initiates a second analysis on the same day, a new folder structure is created and the results from the previous analysis are not overwritten.

## Images Folder

The Images folder contains a subfolder for each lane that has been sequenced. The folders are named using the following convention where the lane number is padded to three digits:

<Sample-ID>\_L<lane number>

If no sample-ID is known, only the lane number is used. For example, L001 contains the images taken in the first lane.

Each lane folder contains a subfolder for each cycle of sequencing. Each image-cycle subfolder contains four images for every tile, one for each of the four bases.

The Image folder naming follows the naming convention C<cycle number>.<version number>. Cycle number is indexed and represents the nth cycle. Version number allows for a cycle to be re-attained if the image acquisition were performed more than once, or the machine paused and a cycle repeated. For example, folders C1.1 and C1.2 would appear if images were acquired twice on the first cycle.

Within each image-cycle subfolder are four tif files for each tile. These files are named using the following convention:

<sample>\_<lane>\_<tile>\_<base>.tif

In the example, s\_1\_67\_g.tif, the "s" is the default sample-ID. Sample-IDs must not contain any underscores. Underscores are used as separators between the different identifiers of the filename to allow easy splitting by any software reading these filenames.

## Data Folder

The Data folder contains a hierarchical structure that consists of the image analysis output folder, then the base calling output folder, and then the sequence alignment output folder.

A new subfolder is generated each time a set of images is processed by the image analysis module (Firecrest). These data are kept in one file per tile for raw intensities and use the extension \_int.txt, and one file per tile for cluster noise and use the extension \_nse.txt.

The Data folder contains a parameters file with multiple records corresponding to each subfolder which has been generated as a result of analyzing sets of images. The detailed information about the image analysis is stored one level above the corresponding data in the directory hierarchy. This allows a user to browse the different results of the image analysis without having to descend into the subfolders.

The parameters file explicitly records which cycle-image folders were used to generate the raw intensities and noise files, and any parameters used. It also records the name of the subfolder and the individual files within it. For a detailed description of the parameters file, see *Parameters* on page 12.



## Image Analysis Folders

Each image analysis subfolder is named using the following convention:

```
C<first cycle>-<last-cycle>_<software><software-
version>_<date>_<user>
```

For example, C1-27\_Firecrest1.8.20\_31-07-2006\_myuser.2 contains the second version of an analysis of cycles 1–27 performed using version 1.8.20 of the Firecrest software, run by the user “myuser” on the 31st of July 2006.

## Base Calling Folders

Each image analysis folder may hold multiple sequence folders with the output of different runs of a base caller package. Each subfolder is named using the following convention:

```
<software><software-version>_<date>_<user>[.<version-number>]
```

For example, the folder name Bustard1.8.8\_08-11-2005\_myuser.3 represents the third run of the Bustard base caller on 8th of November 2005 by the user “myuser.”

Each image analysis folder also holds a parameters file that records any relevant information about the run of the base caller module.

## Run Folder Naming

It is desirable to keep Experiment-Ids (or Sample-ID) and instrument names unique within any given enterprise. You should establish a convention under which each machine is able to allocate Run Folder names independently of other machines to avoid naming conflicts.

The top level Run Folder name is generated using three fields to identify the <ExperimentName>, separated by underscores. For example, YYMMDD\_machinename\_NNNN.

1. The first field is a six-digit number specifying the date of the run. The YYMMDD ordering ensures that a numerical sort of Run Folders places the names in chronological order.
2. The second field specifies the name of the sequencing machine. It may consist of any combination of upper or lower case letters, digits, or hyphens, but may **not** contain any other characters (especially not an underscore). It is assumed that the sequencing instrument is synonymous with the PC controlling it, and that the names assigned to the instruments are unique across the sequencing facility.
3. The third field is a four-digit counter specifying the experiment ID on that instrument. Each instrument should be capable of supplying a series of consecutively numbered experiment IDs (incremental unique index) from the onboard sample tracking database or a LIMS.

A Run Folder named 070108\_instrument1\_0147 indicates experiment number 147, run on instrument 1, on the 8th of Jan 2007. While the date and instrument name specify a unique Run Folder for any number of instruments, the addition of an experiment ID ensures both uniqueness and the ability to relate the contents of the Run Folder back to a laboratory notebook or LIMS.

Additional information is captured in the Run Folder name in fields separated by an underscore from the first three fields. For example, you may want to capture the flow cell number in the Run Folder name as follows:

```
YYMMDD_machinename_XXXX_FCYYY.
```

## File Naming

The Pipeline uses the following format for file naming:

<sample>\_<lane>\_[<tile>\_] [<cycle>\_] [<id>\_] <type>.<filesuffix>

**Table 2** File Naming Components

Component	Description
<sample>	Alphanumeric string
<lane>	Single-digit number identifying a flow cell lane
<tile>	Four-digit number identifying a tile location in a flow cell lane
<cycle>	Two-digit number identifying a sequencing cycle
<id>	Single-digit number to distinguish files; for example, the different reads of a paired-end read
<type>	Alphabetical string identifying the type of content stored in the file
<filesuffix>	Suffix to identify the traditional file type

Example: s\_1\_0010\_01\_2\_clu.txt is a valid filename.

Exceptions:

- ▶ For image (.tif) files, the <tile> location can have less than four digits.
- ▶ For image (.tif) files, the <tile> location may be replaced by two components identifying a row and column in the lane.

## Parameters

The top level Run Folder, the Data Folder and subfolders, and the top level Image folder can all contain a parameters file. This read-only file is intended to contain any parameter data specific to the given level of information held in the folder.

For an example of the parameters file, see *Parameters File Format* on page 89.

## Paired Reads

The simplest way to use paired-read data assumes that you have a single Run Folder containing the images for both reads, with a continuously incremented cycle count.

- ▶ For Genome Analyzer software SCS 1.0 and Pipeline version 0.3 and later, the Pipeline automatically knows where the second read starts.
- ▶ For older versions of Genome Analyzer instrument and analysis software, use the option `--new-read-cycle` to identify the start of the second read. For a description of the `--new-read-cycle` option, see *Command Line Options* on page 21.

An alternative way assumes that both reads of a pair are stored in two separate Run Folders. Specify both folders as arguments to `goat_pipeline.py`. This generates output only in the first Run Folder and the second folder is not touched.

# Calibration and Input Parameters

For an optimal analysis run, the Pipeline needs a number of calibration and input parameters. By default, the Pipeline auto-generates these parameters for each analysis.

Default offsets for runs on the same Genome Analyzer usually do not need to be changed. The Pipeline calculates these parameters automatically and uses them for the corresponding analysis steps.

For samples with biased-base compositions, as encountered in many tag-based or micro RNA applications, auto-calibration does not provide perfect results. For such samples, you need to dedicate one lane of the flow cell to a control sample and use the `--control-lane` command option to generate analysis parameters. For a detailed description, see *Command Line Options* on page 21.

## Image Offsets

There are small pixel offsets among the four differently colored images taken of each tile. These are due to slightly different optical paths for each image. The Pipeline uses offsets to correct for this, and also corrects for linear rescaling of the image.

Each analysis run creates a file called `Data/default_offsets.txt` in the current Run Folder. The `Data/default_offsets.txt` file is used for subsequent analysis of the same run. If the file is located in `Instrument/<instrument>/default_offsets.txt`, the values in the file will be updated during the first run only. File locations are set using the `INSTRUMENT_DIR` variable, as described in *Directory Setup* on page 77.

The `default_offsets.txt` file contains four lines, corresponding to A, C, G, and T respectively, with four values each, using the A image as a reference. The following is an example of a typical `default_offsets.txt` file:

```
# Default offsets
0.00 0.00 0.00000 0.00000
-1.05 -1.62 -0.00017 0.00007
-1.20 -0.47 -0.00143 -0.00142
0.29 -0.92 -0.00159 -0.00142
```

The first two columns in a row correspond to the values of the X and Y offsets of the four images (in pixels).

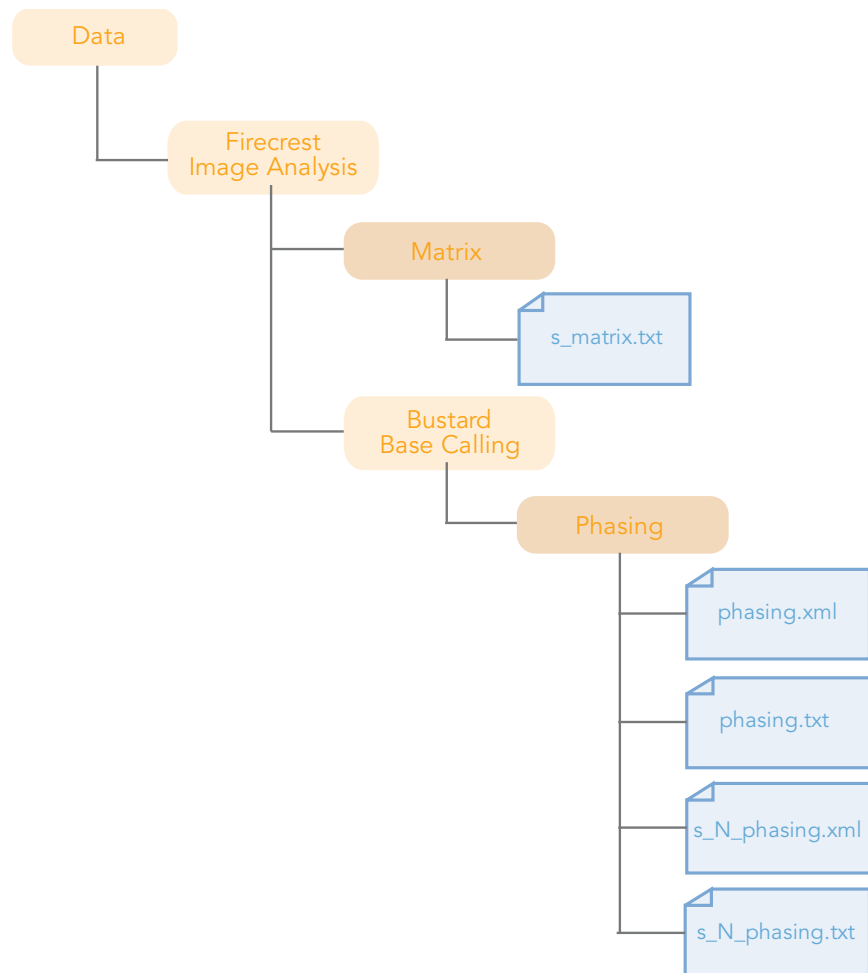
The next two columns indicate scale factors applied to the image.

- ▶ A scale factor of 0 indicates that the image does not need to be rescaled.
- ▶ A scale factor of 0.001 for a 1000 x1000 pixel image indicates that images taken in the corresponding frequency channel tend to be one pixel larger than the reference channel.

## Frequency Cross-Talk Matrix

The Genome Analyzer uses two different lasers to excite the dye attached to each nucleotide. The frequency emission of these four dyes overlaps, so the four images are not independent. As in Sanger sequencing, the frequency cross-talk has to be deconvolved using a frequency cross-talk matrix.

The frequency cross-talk is estimated during the analysis run and captured in a file called `s_matrix.txt`. The `s_matrix.txt` file is located in the Matrix folder as shown in Figure 4.



**Figure 4** Frequency Cross-Talk Matrix and Phasing File Locations

The following is an example of a typical `s_matrix.txt` file:

```

# frequency response matrix definition
> C
> A
> T
> G
1.18  1.29  0.00  0.00
0.18  1.03  0.00  0.00
0.00  0.00  1.43  0.80
0.00  0.00  0.00  0.71
  
```

The lines starting with a greater than symbol (“>”) specify the order of the rows and columns in terms of the bases they represent.

The matrix elements show how the A, C, G, and T dyes/nucleotides (columns) cross-talk into the A, C, G, and T channels. A normal matrix should be diagonally dominant (diagonal elements tend to be the largest values) with the exception of the top-left and bottom-right corners (A/C and G/T cross-talk respectively). These are not as well-separated due to the fact that both corresponding dyes are excited by the same laser.

## Phasing/Prephasing Estimates

Depending on the efficiency of the fluidics and the sequencing reactions, a small number of molecules in each cluster may run ahead (prephasing) or fall behind (phasing) the current incorporation cycle. This effect can be mitigated by applying corrections during the base calling step.

The phasing estimates are produced before a run of the base caller module and captured in a file called phasing.xml. The phasing.xml file is located in the Phasing folder as shown in Figure 4.

As the estimation uses statistical averaging over many clusters and sequences to estimate the correlation of signal between different cycles, the phasing estimates tend to be more accurate for tiles with larger numbers of clusters and a mixture of different sequences. Samples containing only a small number of different sequences do not produce reliable estimates.

## Sample Information

Depending on the application, a reference genome may be supplied for the read sequences to be aligned against.

## Alignment Algorithms

The Pipeline provides two alignment algorithms: PhageAlign and ELAND.

- ▶ PhageAlign performs an exhaustive alignment and always finds the best match but is very slow.
- ▶ Efficient Large-Scale Alignment of Nucleotide Databases (ELAND) is very fast and used to match a large number of reads against the human genome with no more than two errors in the first 32 bases.

ELAND searches a set of large DNA files for a large number of short DNA reads allowing up to 2 errors per match. This description is based on the following definitions:

- ▶ **Large**—Human genome size and above, including small genomes
- ▶ **Large number**—At least eight million on a PC with 1 GB of RAM
- ▶ **Short**—32 bases or less

ELAND is much faster than PhageAlign but will only detect matches with two differences or fewer from your reads. This means that ELAND is less sensitive than PhageAlign, which will always find a best match (although possibly not a unique one) for your reads. Consider the following points when using ELAND:

- ▶ If your data is noisy, not all of it is going to align. If this happens to a significant proportion of your data, then it is possible that your data is too noisy to get good results.
- ▶ Error rates based on ELAND output underestimate the true error rate. Since reads with two or more errors in the first 32 bases do not get aligned, they do not contribute to the calculation.



## Chapter 3

# Running the Analysis

### Topics

- 18 Introduction
- 18 Starting the Genome Analyzer Pipeline Software
- 19 Running a Standard Analysis
  - 19 Specifying the IPAR Folder
  - 20 Parallelization Switch
  - 20 Nohup Command
- 21 Command Line Options
  - 21 General Options
  - 22 GOAT Options
  - 22 GOAT and Bustard Options
  - 23 Paired Reads
  - 23 IPAR Analysis
  - 24 Makefile Targets

## Introduction

This section describes the standard analysis run and command line options.

The standard invocation of the Pipeline assumes that you are performing image analysis, base calling, and sequencing alignment on a set of images in the Run Folder. It also assumes that the images are organized in a standard Run Folder directory structure as described in *Run Folder Structure* on page 9.

To successfully initiate image analysis, you need four images for each tile, for each cycle, and a parameters (.params) file in the Run Folder.

## Starting the Genome Analyzer Pipeline Software

Although several different software programs are involved in an analysis run, a single command `goat_pipeline.py` will start the Pipeline automatically, and then trigger the launch of subsequent utilities.

This is the standard invocation of the Pipeline. Arguments contained in brackets [ ] are optional.

```
/path/Pipeline/Goat/goat_pipeline.py [--cycles=1-25|auto] [--tiles=s_1,s_2_0003,...]
  [--matrix=mymatrix.txt|auto|auto<n>] [--offsets=/path/default_offsets.txt|auto]
  [--phasing=0.01|auto|auto<n>] [--prephasing=0.01]
  [--directory=/path/C1-14_Firecrest1.8.20_01-08-2006_user] [--make]
  [--GERALD=/path/config.txt] [--control-lane=5]
  <run-folder-directory>|<IPAR directory> [<run-folder-directory2>]
```

Some of the arguments above have sample values displayed. The only compulsory argument is the path to the Run Folder that is to be analyzed. The path can also point to any folder containing tiff images that are to be analyzed. Alternatively, you can provide a space-separated list of TIFF filenames.



## Running a Standard Analysis

A standard analysis consists of calling the `goat_pipeline.py` script to generate an analysis directory using the “make” command, and then executing the “make” command.

Start a standard analysis run using the following command format:

```
Pipeline/Goat/goat_pipeline.py
  [--GERALD=<configfile>] [--make] <run-folder>
```

1. Type the following command to run a check on the Run Folder, report all detected folders and parameters files, and fill in any missing configuration options.

```
Pipeline/Goat/goat_pipeline.py
  --GERALD=/data/070813_ILMN-1_0217_FC1234/config.txt
  /data/070813_ILMN-1_0217_FC1234
```

Illumina recommends running this script before generating the makefile to check for data integrity and consistency. It scans all the images folders and prints diagnostic output about the images and parameters files. No files or directories are modified on the data drive as a result of this command.

2. Add `--make` to the command listed above to create an analysis directory in the Run Folder. If you specify the `--GERALD` option, you will create the GERALD analysis folder and the corresponding makefile.

```
Pipeline/Goat/goat_pipeline.py
  --GERALD=/data/070813_ILMN-1_0217_FC1234/config.txt
  --make /data/070813_ILMN-1_0217_FC1234
```

3. Change to the newly generated directory (for example, `/data/070813_ILMN-1_0217_FC1234/Data/C1-26_Firecrest`) and type the “make recursive” command. This command starts the actual analysis.

```
make recursive
```

For more information on “make recursive,” see *Makefile Targets* on page 24.

The primary outputs are the sequences read with per-base quality values and, if alignment was performed, the alignments. These files can be found in the GERALD folder. The output files containing data statistics and histograms, used for quality control, can also be found in the GERALD folder.

A new output directory is created each time you rerun the analysis, so there is no need to remove any previous analysis files.

### Specifying the IPAR Folder

The Integrated Primary Analysis and Reporting (IPAR) module performs image analysis and lets you assess the quality of run data in real-time on the Genome Analyzer. IPAR analysis generates output results which are saved in the `<RunFolder>\Data\<IPAR Folder>` created for the run in progress.

Instead of specifying the Run Folder, you can specify the IPAR folder that was produced by the IPAR module. The advantage to specifying the IPAR folder is that the Pipeline will start from base calling using the image analysis results generated by IPAR. Subsequently, the time for complete analysis of an experiment is considerably less. In this case, a new folder for image analysis is not created and the additional Pipeline output is located in the IPAR folder and related subfolders.

The following prerequisites must be met:

- ▶ You must be running Pipeline version 1.0. Earlier versions of the Pipeline software are not compatible with IPAR.
- ▶ The experiment Run Folder containing the IPAR image analysis results is copied to the off-line server where Pipeline is running.
- ▶ The parameters file for the experiment is copied in \RunFolder\Data.

If you are using all mechanisms for data transfer provided by Illumina, the second and third prerequisites will always be met.

## Parallelization Switch

If your system supports automatic load-sharing to multiple CPUs, you can parallelize the analysis run to <n> different processes by using the "make" utility parallelization switch.

```
make recursive -j n
```

For more information on parallelization, see *Using Parallelization* on page 93.

## Nohup Command

You can use the Unix nohup command to redirect the standard output and keep the "make" process running even if your terminal is interrupted or if you log out. The standard output will be saved in a nohup.out file and stored in the location where you are executing the makefile.

```
nohup make recursive -j n &
```

The optional "&" tells the system to run the analysis in the background, leaving you free to enter more commands.

# Command Line Options

You can invoke the `goat_pipeline.py` and `bustard.py` scripts with a number of optional command line arguments.

## General Options

Any of the following general options can be included in any order on a single command line.

### `--make`

The `--make` command creates the analysis directory and a makefile in the relevant analysis directory. You can start the analysis by changing to the directory and typing "make." If this option is omitted, the Pipeline will not write any information to your Run Folder.

### `--new-read-cycle=<cycle>`

Use this command to start a new read in a paired-end run. The calculation of the matrix correction and the application of the phasing correction will be reset at the specified cycle.

### `--GERALD=<config.txt>`

Use this command to start the GERALD makefile generator after the Bustard folder is created. You can specify multiple GERALD files by repeating the option with different configuration file names. For each GERALD configuration file specified, a separate GERALD subfolder is generated (under the same Bustard folder) with that configuration. For more information on the GERALD configuration file, see *GERALD Configuration File* on page 35.

### `--tiles=<tile>|<lane>[,<tile>|<lane>,...]`

Use this command to select certain tiles for analysis. For example, specifying `--tiles=s_1,s_2_01,s_3_0001,s_5_0002` selects all tiles in lane 1, all tiles starting with "01" in lane 2, position 1 in lane 3, and position 2 in lane 5.

You can also specify certain tiles for analysis from every lane. For example, specifying `--tiles=_0010,_0020` selects only tiles 10 and 20 from every lane.

### `--cycles=<cycle>[-<cycle>[,<cycle>[-<cycle>...]]]`:

Use this command to select certain cycles for analysis. For example, use `--cycles=3-31` to include only cycles 3 through 31 in the analysis.



If you skip cycles in the middle of a read, you cannot use ELAND to align the data.

Using the value "auto" tells the Pipeline to automatically select the lowest number of cycles present in any of the tiles and to make sure that all tiles have equal read lengths, regardless of the state of data acquisition/mirroring.

**--compression=<method>**

Use "--compression" to reduce the size of the Firecrest output. Allowed values are "none," "gzip" (the default), and "bzip2."

In the Pipeline version 0.3 and later, the intensity files are compressed by default. For previous versions, you must specify "--compression=none" on the command line.

**GOAT Options**

Use the following options with the goat\_pipeline.py script.

**--nobasecall**

Use --nobasecall to skip the base calling step in the analysis.

**--offsets=<filename>|autoldefault**

Use --offsets=<filename> to specify a certain default offset file. If no offset file is specified, the Pipeline will create one in the Instruments folder.

**GOAT and Bustard  
Options**

Use the following options with goat\_pipeline.py and bustard.py scripts.

**--control-lane=<n>**

Use this command to select a lane <n> that is to be used to estimate phasing and matrix correction for all other lanes. This option is synonymous with --phasing=auto<n> --matrix=auto<n>. Control lanes are necessary for samples with skewed base compositions.

**--matrix=<filename>|autolauto<n>|lane**

Use the --matrix command to specify the frequency cross-talk matrix file, where filename refers to the path of the matrix file.

If no matrix is specified, or if you set the value to the default behavior "auto," the Pipeline auto-generates the matrix. A value of auto<n>, where <n> is a lane number between 1 and 8, is analogous to the --phasing=auto<n> option and allows the matrix estimation to be derived from only one lane.

**--phasing=<x>|autolauto<n>**

Use the --phasing command to apply a particular phasing correction. If you set the value to the default behavior "auto," the Pipeline auto-generates the phasing and prephasing values.

A value of auto<n>, where <n> is a lane number between 1 and 8, uses the automated phasing estimates from the corresponding lane. This is useful for samples with an uneven base composition (such as in gene expression), for which the current phasing estimator does not work reliably and phasing needs to be estimated from a single control lane.

You can specify a phasing value directly. For example, --phasing=0.01 indicates a phasing correction with a rate of 1% per cycle (1% of molecules in a cluster fall behind the other molecules). In this case, the option is normally combined with the --prephasing option.

**--prephasing=<x>**

Use the `--prephasing` command to apply a particular correction for prephasing. For example, using `--prephasing=0.01` sets a correction for prephasing with a prephasing rate of 1% per cycle.

The command `--prephasing=auto` is not recognized. Use `--phasing=auto` instead. By default the Pipeline autogenerates phasing estimates.

**Paired Reads**

The following additional variations on the `goat_pipeline.py` and `bustard.py` options are supported for paired reads.

**--phasing=<read>:value, --phasing=<read>:<read>**

Use this command to specify phasing options for one specific read of a pair.

The following example uses the default phasing option for read 1 but uses base phasing estimates from lane 5 for read 2:

```
--phasing=1:auto --phasing=2:auto5
```

The following example uses the phasing estimate for the second read and applies it to both read 1 and read 2:

```
--phasing=1:2
```

**--matrix=<read>:value, --matrix=<read>:<read>**

Use this command to specify matrix options for one specific read of a pair. This is analogous to the phasing options listed above.

**IPAR Analysis**

Image analysis data generated with IPAR can be processed by the Pipeline software. Pipeline performs base calling after calculating the cross-talk matrix, phasing, and prephasing values.

Specify the IPAR folder as an argument to `goat_pipeline.py` instead of the Run Folder:

```
goat_pipeline.py <IPAR folder>
```

If the Run Folder is specified instead of the IPAR folder, the Pipeline will restart image analysis.

1. Invoke the `goat_pipeline.py` script to generate the Pipeline makefiles and analysis directory.

```
<path to Pipeline>/Goat/goat_pipeline.py <IPAR Folder>
--make
```

You must specify the entire path to the IPAR analysis folder. A typical IPAR 1.0 folder and path may be as follows:

```
data/070813_ILMN-1_0217_FC1234/Data/IPAR_1.0
```

The IPAR version number in the folder name is the version number of the IPAR system generating the analysis.

All standard Pipeline variables are available for use with the IPAR folder. For example, you can use `--GERALD=config_file.txt` to specify alignment information.

2. To execute the makefiles, navigate to the IPAR folder and use "make recursive."

```
cd <Run Folder>/Data/<IPAR Folder>
```

```
make recursive
```

Using “make all” will generate the matrix file in the IPAR directory, but will not initiate base calling.

## Makefile Targets

Both `goat_pipeline.py` and `bustard.py` scripts generate makefiles in the relevant image analysis and base caller directories that allow the complete analysis to be run by GNU Make. The makefiles have the following advantages:

- ▶ Not all of the analysis needs to be run immediately.
- ▶ On a multiprocessor system or cluster, the analysis can easily be parallelized by specifying the “-j” option for “make.”
- ▶ In case of any failure or interruption during an analysis run, the run can easily be restarted at the last point.

The following optional targets are used with the “make” command.

### all

All is the default makefile target. It runs the complete analysis in the current directory (image analysis or base caller).

### <sample>\_<lane>

This target analyzes all tiles in a lane. For example, use `make s_1 s_2` to analyze lanes 1 and 2.

### <sample>\_<lane>\_<tileindex>

This target analyzes a specific tile only. For example, use `make s_1_0007` to analyze lane 1, tile 7. This target is incompatible with auto-generated matrices and phasing estimates.

### \_<tileindex>

This target analyzes all tiles with the given index from any lane and is useful for analyzing randomly chosen subsets of a tile. For example, use `make _0020 _0040 _0060` to analyze tiles with indices 20, 40, 60 in all lanes.

This target is currently incompatible with auto-generated matrices and phasing estimates.

### -j <n>

This parallelization switch can be used with the “make” command to execute the Pipeline run in parallel over <n> number of processor cores. For a description of parallelization, see *Using Parallelization* on page 93.

### clean

This target removes all analysis output files. You would use “make clean” when you are low on disk space.



**CAUTION**

Using “make clean” removes all analysis results from the folder where the command is executed. Use with care.

## recursive

This target performs the analysis in the current directory and in all available subdirectories. Use this target to start a complete end-to-end analysis run from image analysis or base calling to sequence alignment using a single command.

The following example starts recursive full analysis:

```
make recursive
```

Specify the target by setting the TARGET environment variable. The following example removes all analysis results from ALL subfolders:

```
make recursive TARGET=clean
```

The recursive option is not compatible with tile and lane-specific targets.

## compress

This target uses gzip to apply a loss-less compression to the output files after an analysis run. This significantly reduces the size of the analysis folders. Typically, the Firecrest and Bustard folders are reduced to 1/3 and 1/4 of their original size.

In the compressed state, no further analysis is possible. The folder must be uncompressed in order to reanalyze it.

## uncompress

This target uncompresses a folder that has previously been compressed and returns it to its original state.

## compress\_images

This target uses bzip2 to compress the image data in the Images folder. This can take a significant amount of time, but reduces the size of the Images directory to about 60% of its original size.

In the compressed state, no further analysis is possible. The folder must be uncompressed in order to reanalyze it.

## uncompress\_images

This target uncompresses the Images folder that has previously been compressed and returns it to its original state.







## Chapter 4

# Using GERALD

### Topics

- 28 Introduction
- 29 GERALD Parameters
  - 29 ANALYSIS Variables
  - 30 Analysis Parameters
  - 31 Filtering Parameters
  - 31 USE\_BASES Option
  - 32 Lane-by-Lane Parameters
  - 33 FORCE Option
  - 33 Rerunning the Analysis
  - 33 Contaminant Filtering
- 35 GERALD Configuration File
  - 36 Lane-Specific Options
  - 36 Optional Parameters
  - 37 Paired-End Analysis Options
- 38 Preparing the Reference Genome
- 40 ELAND Alignments
  - 41 Missing Bases in ELAND
  - 41 Using ANALYSIS eland\_tag
  - 42 Using ANALYSIS eland\_extended
  - 43 Using ANALYSIS eland\_pair

## Introduction

GERALD is the module that performs sequence alignments, visualization, produces statistics, and analysis output in a series of diagnostic QC plots and summary tables. These are presented in the form of html pages found in the GERALD output folder.

GERALD is usually run automatically as part of an overall Pipeline analysis but can also be run independently. For more information, see *Running GERALD as a Standalone Program* on page 67.

As a result of running the GERALD.pl script, a new directory is created and named using the format GERALD\_DD-MM-YYYY\_user where the date is the current date and user is your computer login. If you want to rerun the analysis and change parameters, you can rerun GERALD with new parameters. A new directory will be created and no information will be overwritten.

GERALD uses multiple analysis parameters. Therefore, it is recommended to include the parameters in a configuration file and provide that file as input to GERALD.

You can define GERALD analysis parameters in the configuration file or in the command line. Command line arguments take precedence over parameters set in the configuration file. For a full description of analysis parameters and variables, see *GERALD Parameters* on page 29.

The following is an example of a GERALD invocation using a configuration file and command line arguments:

```
GERALD.pl config.txt --EXPT_DIR
/data/070813_ILMN-1_0217_FC1234/Data/C1-
27_Firecrest1.9.0_23-08-2007-user/Bustard1.9.0_23-
08-2007_user/
--FORCE --GENOME_DIR /data/Genomes --GENOME_FILE
BAC_plus_vector.fa
```

This section describes the GERALD parameters, analysis variables, configuration file options, and ELAND alignments.

## GERALD Parameters

GERALD can be run in various analysis modes. Your analysis can be customized by specifying variables, parameters, and options.

### ANALYSIS Variables

Set the ANALYSIS variable to define the type of analysis you want to perform for each lane. The various analysis modes include default, sequence, eland, eland\_extended, eland\_tag, eland\_pair, none, and monotemplate. You can mix and match analyses between lanes.

For all modes, except ANALYSIS none, you will get a sequence output file (s\_N\_sequence.txt) for each lane.

**Table 3** ANALYSIS Variables

Variable	Alignment Program	Application	Description
ANALYSIS eland_extended	ELAND	Single reads	<p>An improved version of ANALYSIS eland for analyzing single-read data.</p> <ul style="list-style-type: none"> <li>• Better handling of &gt; 32 base reads</li> <li>• Each alignment is given a confidence value based on its base quality scores</li> <li>• A single file of sorted alignments is produced for each lane</li> </ul> <p>For a detailed description, see <i>Using ANALYSIS eland_extended</i> on page 42.</p>
ANALYSIS eland_pair	ELAND	Paired reads	<p>Aligns paired-end reads against a target using ELAND alignments. A single-read alignment is done for each half of the pair, and then the best-scoring alignments are compared to find the best paired-read alignment. For a detailed description, see <i>Using ANALYSIS eland_pair</i> on page 43.</p>
ANALYSIS eland_tag	ELAND	Gene Expression	<p>Aligns reads to a non-redundant reference set of separate sequence tags and produces exact matches. For additional information, see <i>Using ANALYSIS eland_tag</i> on page 41.</p>
ANALYSIS monotemplate	PhageAlign	Single reads	<p>Aligns to a tag set using PhageAlign. Setting the parameter 6:ANALYSIS monotemplate performs monotemplate analysis for lane 6, where you can expect each read to be one of a small number (20 or less) of known template sequences. The reads are aligned using PhageAlign in tag mode, which treats each line of the reference sequence as a separate tag. No coverage plots will be produced since they are not relevant here. Some monotemplate-specific output is produced instead.</p>
ANALYSIS sequence	None	Single reads Paired reads	<p>Produces one file of sequence output per lane with no alignment. Setting the parameter 6:ANALYSIS sequence produces a file named s_6_sequence.txt. This file contains all sequences in a lane of a flow cell in an exportable format.</p>

**Table 3** ANALYSIS Variables (Continued)

Variable	Alignment Program	Application	Description
ANALYSIS sequence_pair	None	Paired reads	Produces two files of sequence output per lane, with no alignment. For example, s_1_1_sequence.txt and s_1_2_sequence.txt contain sequence output, one file for each half of the read pair.
ANALYSIS none	None	Any application	Omits the indicated lane from the analysis. Setting the parameter 8:ANALYSIS none ignores lane 8.
ANALYSIS default	PhageAlign	Single reads	Aligns each read against a reference sequence using PhageAlign. This mode is suitable only for small genome references.
ANALYSIS eland	ELAND	Single reads	Aligns each read against a large reference sequence using ELAND. Setting the parameter 6:ANALYSIS eland runs an ELAND whole-genome analysis for lane 6. You need to use ELAND if your reference sequence exceeds 1 MB in size. No coverage files will be generated. For more information on ELAND, see <i>ELAND Alignments</i> on page 40.
ANALYSIS expression	PhageAlign	Gene Expression	Aligns reads to a tag set using PhageAlign. This analysis mode is deprecated in favor of ANALYSIS eland_tag.

**Analysis Parameters** The content of the output file is determined by the following analysis parameters.

**Table 4** Analysis Parameters

Parameter	Description
USE_BASES	Use this parameter to identify bases to be used for alignment analysis. The USE_BASES string uses an asterisk (*) to indicate "fill up the read as far as possible with the preceding character." If USE_BASES all is set, all sequenced bases will show up in the analysis results. Otherwise, only cycles which have a Y at the corresponding position in the USE_BASES string will appear in the results. For a detailed description of USE_BASES syntax, see <i>USE_BASES Option</i> on page 31.
QF_PARAMS	Use this parameter if you want to use filtering different than the default filter. Set QF_PARAMS '(1==1)' to pass all of them. For information on default filtering, see <i>Filtering Parameters</i> on page 31.

Table 4 Analysis Parameters (Continued)

Parameter	Description
SEQUENCE_FORMAT	<p>This parameter specifies what format to use for data export in the s_N_sequence.txt file. Allowed values are --fasta, --fastq, or --SCARF.</p> <ul style="list-style-type: none"> <li>• <b>fasta</b>—This format is widely used but does not contain quality scores.</li> <li>• <b>fastq</b>—This format is an adaption of the fasta format that contains quality scores. However, the fastq format is not completely compatible with the fastq files currently in existence, which is read by various applications (for example, BioPerl). Because a larger dynamic range of quality scores is used, the quality scores are encoded in ASCII as 64+score, instead of the standard 32+score. This method is used to avoid running into non-printable characters.</li> <li>• <b>SCARF (Solexa compact ASCII read format)</b>—This easy-to-parse text based format, stores all the information for a single read in one line.</li> </ul>
QUALITY_FORMAT	<p>Allowed values are --numeric and --symbolic.</p> <ul style="list-style-type: none"> <li>• <b>--numeric</b> outputs the quality values as a space-separated string of numbers.</li> <li>• <b>--symbolic</b> outputs the quality values as a compact string of ASCII characters. Subtract 64 from the ASCII code to get the corresponding quality values. Under the current numbering scheme, quality values can theoretically be as low as -5, which has an ASCII encoding of 59=';';</li> </ul>

## Filtering Parameters

GERALD uses filtering to remove low-quality base calls. By default, a filter discards all clusters with a ratio less than or equal to 0.6 between the highest and the sum of the highest two intensities for the first 12 cycles.

Alternative to the default filter called CHASTITY, the filters PURITY and NEIGHBOUR can be used:

- ▶ **CHASTITY**—The ratio of the brightest intensity over the sum of the brightest and second brightest intensities per base
- ▶ **PURITY**—The ratio of the brightest intensity over the sum of all of the four intensities per base
- ▶ **NEIGHBOUR**—Distance in pixels to the nearest neighboring cluster

## USE\_BASES Option

The USE\_BASES option identifies which bases of a full read produced by a sequencing run should be used for the alignment analysis. A fully expanded USE\_BASES value is a string with one character per sequencing cycle but more compact formats can be used as described in Table 5 on page 32. Each character in the string identifies whether the corresponding cycle should be aligned. The following notation is used:

- ▶ A lower-case "n" means ignore the cycle.  
The first base could be ignored because it is part of the sequencing primer. The last base could be ignored because it is not corrected for prephasing and may have higher error rates.
- ▶ An upper-case "Y" means use the cycle for the alignment.
- ▶ A comma (,) denotes a read boundary used for multiple reads.

- ▶ An asterisk (\*) means “fill up the read as far as possible with the preceding character.”
- ▶ A number means that the previous character is repeated that many times. Unspecified cycles are set to “n” by default. If USE\_BASES is not specified at all, every cycle is used for the alignment.

The following table describes examples of USE\_BASES options.

**Table 5** USE\_BASES Options

Option	Definition
USE_BASES nYYY	Ignore the first base and use bases 2–4.
USE_BASES Y30	Align the first 30 bases.
USE_BASES nY30	Ignore the first base and align the next 30 bases.
USE_BASES nY30n	Ignore the first base, align the next 30 bases, and ignore the last base.
USE_BASES nY*n	Ignore the first base, perform a single read, and ignore the last base. The length of read is automatically set to the number of sequencing cycles minus two.
USE_BASES nY*,nY*	Ignore the first base of each read and perform a paired read, resulting in the length of each read being set to the number of sequencing cycles associated with it minus one. The two reads do not need to be of the same length.
USE_BASES nY*	When used with ANALYSIS eland_pair, this is an abbreviation for USE_BASES nY*,nY*. When used with a single-read analysis mode, this means ignore the first base and perform a single-read.
USE_BASES all	Use all bases.

### Lane-by-Lane Parameters

You can set the analysis parameter and other parameters on a lane-by-lane basis. You will need to do this for any parameters specific to the analysis of a particular lane.

**Table 6** Lane-by-Lane Parameters

Option	Definition
6:ELAND_GENOME/directory/genome	Specify the name of the file containing the reference sequence for lane 6.
67:ELAND_GENOME/directory/genome	Specify the name of the file containing the reference sequence to use for lanes 6 and 7.

**Table 6** Lane-by-Lane Parameters (Continued)

Option	Definition
<pre>3:QF_PARAMS=' (( NEIGHBOUR&gt;=   5.0 )&amp;&amp;( PURITY&gt;=0.7 )&amp;&amp;( (   TILE!=4 )    ( X_COORD&gt;50 ) )   )'</pre>	<p>Set the quality filtering parameter, QF_PARAMS, on a lane-by-lane basis.</p> <p>For example, if the clusters near the left edge of tile 4 of lane 3 look questionable, use this command to set the filter.</p> <p>You can use any Boolean Perl expression as a filter parameter. The variable names are aliases to fields in a tab-separated text file.</p> <p>The best way to filter out individual tiles is to set BAD_TILES to be a list of the tiles you want to filter. See <i>Optional Parameters</i> on page 36 for an example of the BAD_TILES parameter.</p>

## FORCE Option

The FORCE option creates GERALD directories and makefiles. Without the FORCE option, GERALD will not create any directories and files and only operates in a diagnostic mode. You must specify this option to generate the GERALD analysis folder and subsequently run the analysis.

## Rerunning the Analysis

The config.txt file used to generate an analysis is copied to the analysis folder so it can be used by GERALD if a reanalysis of the same data is required. To change parameters and rebuild the analysis, modify the configuration file and run the following command:

```
GERALD.pl config.txt --FORCE
```

By adding the OUT\_DIR option, you can force GERALD to overwrite an existing makefile. This way you can modify the analysis without directly editing the makefile.

## Contaminant Filtering

Contaminant filtering can be used with the variables ANALYSIS default and ANALYSIS eland. However, most of the time it is not needed.

GERALD attempts to filter contaminant sequences in a rigorous way by comparing the alignment of each read against the data versus the best alignment to the contaminant sequences. A one-sequence-per-line ASCII file is expected, with each sequence being at least READ\_LENGTH bases in length.

To switch contaminant filtering on, specify the name of the file containing contaminant sequences in CONTAM\_FILE. It is assumed that the CONTAM\_FILE is located in GENOME\_DIR. If it is not, specify the location in CONTAM\_DIR.

## Building an SRF Archive

With version 1.0, the Pipeline is distributed with a modified version of io\_lib and allows the generation of SRF archives. This is done by adding the following line in the config.txt file:

```
1:SRF_ARCHIVE_REQUIRED yes
```

## Including the Quality Calibration Scores

The modified version of io\_lib distributed with the Pipeline also supports the inclusion of the quality calibration scores into the SRF archives. Add the following line in the config.txt file:

```
1:SRF_QCAL yes
```

## Extracting the Data from the Archive

The data is extracted from the archive by an `io_lib` utility called `srf2illumina`. The standard distribution of `io_lib` is able to extract the data from the archive with the exception of the quality scores. The modified version distributed with the Pipeline is required to include the quality scores.



# GERALD Configuration File

This section describes a typical GERALD configuration file that uses the current features and parameters.

As part of the creation of the GERALD output folder, the GERALD configuration file specified (whether using the `--GERALD` option of `goat_pipeline.py` or directly as the argument to `GERALD.pl`) is copied to the GERALD output folder using the filename `config.txt`. Some sites use standard configuration files, which may be stored in a central repository.

The GERALD configuration file specifies what analysis should be done for each lane, which GERALD translates into a makefile. The makefile specifies exactly what commands should be executed to carry out the requested analysis.

**Table 7** GERALD Configuration File Parameters

Parameter	Definition
<code>EXPT_DIR</code> <code>data/070813_ILMN-1_0217_FC1234/Data/C1-27_Firecrest1.9.0_23-08-2007-user/Bustard1.9.0_23-08-2007_user/</code>	Provide the path to the experiment directory, if not specified on command line or auto-completed by <code>goat_pipeline.py</code> .
<code>OUT_DIR</code> <code>/home/user/Test4</code>	Indicate the output directory, if other than a new GERALD folder inside of the <code>EXPT_DIR</code> folder.
<code>USE_BASES</code> <code>nY*n</code>	Ignore the first and last base of the read. The <code>USE_BASES</code> string contains a character for each cycle. <ul style="list-style-type: none"> <li>• If the character is "Y," the cycle is used for alignment.</li> <li>• If the character is "n," the cycle is ignored.</li> <li>• Wild cards (*) are expanded to the full length of the read.</li> </ul> For a detailed description of <code>USE_BASES</code> syntax, see <i>USE_BASES Option</i> on page 31.
<code>ELAND_GENOME</code> <code>/home/user/Genomes/Eland/BAC_plus_vector/</code>	Specify the genome reference for alignment with ELAND.
<code>GENOME_DIR</code> <code>/home/user/Genomes</code>	Specify where the genome file is located.
<code>ANALYSIS</code> <code>eland</code>	Align against a genomic sample and allow alignments to arbitrary positions of the provided reference.
<code>READ_LENGTH</code> <code>25</code>	This parameter is no longer used with software version 0.3 and later. Specify the read length for the experiment. This is the read length used for the alignments, not the sequenced read length. Consequently, the value has to be less than or equal to the sequence length. It is useful to force the wild card expansion of <code>USE_BASES</code> to a predefined value.

## Lane-Specific Options

The following table describes the lane-specific parameters in a GERALD configuration file.

**Table 8** GERALD Configuration File Lane-Specific Options

Parameter	Definition
<code>7:USE_BASES nY20</code>	Align only 20 cycles for lane 7, starting with the second cycle.
<code>567:ANALYSIS sequence</code> <code>567:USE_BASES all</code>	Output sequence information for lanes 5, 6, and 7 only (no alignments are performed).
<code>8:ANALYSIS none</code>	Omit lane 8, which only contains primers.
<code>3:QF_PARAMS</code> <code>' (( NEIGHBOUR &gt;= 3.0 ) &amp;&amp; ( CHASTITY &gt;= 0.7 ) &amp;&amp; ( X_COORD &gt; 50 ) ) '</code>	Filter parameters using the default (CHASTITY >= 0.6). This example includes only those clusters with a separation of at least three pixels, with a CHASTITY filtering greater than or equal to 0.7, and an X coordinate greater than or equal to 50.

## Optional Parameters

The following table describes the optional parameters in a GERALD configuration file.

**Table 9** GERALD Configuration File Optional Parameters

Parameter	Definition
<code>EMAIL_LIST user@example.com</code> <code>user2@example.com</code> <code>EMAIL_SERVER mailserver</code> <code>EMAIL_DOMAIN example.com</code>	<b>[Optional]</b> Send notification to the user at end of an analysis run. For more information on email notification, see <i>Setting Up Email Reporting</i> on page 75.
<code>WEB_DIR_ROOT file://</code> <code>server.example.com/share</code>	<b>[Optional]</b> Include hyperlinks with a specific prefix to the Run Folder.
<code>BAD_TILES s_1_0001 s_2_0003</code>	Identify bad tiles. These tiles will be aligned but excluded from coverage.
<code>POST_RUN_COMMAND /yourPath/</code> <code>yourCommand yourArgs</code>	Allows user-defined scripts to be run after all GERALD targets have been built.

## Paired-End Analysis Options

The following table describes the paired-end analysis options in a GERALD configuration file.

**Table 10** GERALD Configuration File Paired-End Analysis Options

Parameter	Definition
<code>ANALYSIS eland_pair</code>	Use the paired-end alignment mode of ELAND to align paired reads against a target.
<code>USE_BASES Y*,nY*n</code>	Use all bases on the first read and ignore the first and last base of the second read.
<code>6:USE_BASES nY25</code>	Ignore the first base on both the first and second read; use 25 bases each and ignore any other bases.

For more information on USE\_BASES syntax, see *USE\_BASES Option* on page 31.

## Preparing the Reference Genome

Several of the GERALD analysis modes (namely `eland`, `eland_extended`, `eland_pair`, and `eland_tag`) make use of the ELAND alignment program to align the reads produced by the Pipeline against a set of reference sequences. Before you can run an analysis that uses ELAND, you need to obtain the reference sequences you wish to align against in fasta format and convert or “squash” them into the format that ELAND can read. This is done by running a program `squashGenome` that is provided as part of the Pipeline installation.

The outcome of the squashing process is a folder containing a set of files that encode the reference sequences in a 2-bits-per-base binary format that is not human readable. Squashing only needs to be done once for each set of reference sequences you are interested in aligning against. For example, if you were doing some mouse and some human sequencing, you might create a folder containing a squashed version of the mouse genome and another folder containing a squashed version of the human genome (you probably do not want to squash both genomes into the same folder). Once created, a squashed folder can be copied between machines or placed on a shared drive, so as to be accessible from multiple machines. You specify the path of this folder as a parameter `ELAND_GENOME` when creating the configuration file for any analysis that involves ELAND.

See *Using ANALYSIS eland\_tag* on page 41 for additional instructions on preparing a set of sequence tags for use as a reference sequence in `eland_tag` mode.

The fasta file format is very well known. Here is an example:

```
>chromosome:NCBI36:X:1:154913754:1
CTAACCTAACCTAACCTAACCTAACCTAACCTAACCTCTGAAAGTGG
ACCTATCAGCAG
GATGTGGGTGGGAGCAGATTAGAGAATAAAAGCAGACTGCCTGAGC
CAGCAGTGGCAACC
```

Please note that the names of the entries in any fasta files to be squashed cannot contain spaces. In `eland_pair` and `eland_extended`, the names of the entries cannot contain spaces, commas, or colons.

1. You must first create an empty folder for the squashed files to go into.
2. Go to the location of the fasta format reference sequence files and enter the following command:

```
<Pipeline>/Eland/squashGenome <path>/myGenome
  fastaFile1.fa [fastaFile2...]
```

where `<Pipeline>` denotes the full path of the Pipeline installation location and `<path>` denotes the full path to the folder `myGenome` you created in step 1. This will cause files `fastaFile1.fa.2bpb` and `fastaFile1.fa.vld` to be created in folder `myGenome`.

Prior to Pipeline version 0.3, there was a restriction of a single entry per fasta file for the reference sequences. For Pipeline version 0.3 and later, this restriction has been removed and fasta files with multiple entries can be squashed.

For reasons of efficiency, ELAND thinks of the reference sequence as being in “blocks” of 16 MB, of which there can be at most 240. This limits the total length of DNA that ELAND can match against in a single run.

In a single ELAND run you can match against:

- ▶ One file of at most  $240 \times 16 = 3824$  MB
- ▶ 239 files, each up to 16 MB in size
- ▶ Something in between, such as 24 files of up to 160 MB each. (The NCBI human genome will fit.)

**NOTE**

ELAND does not check that the directory of squashed genomes it is matching against exceeds these limits.

Each file in the reference sequence must take up at least one block, so if you have a large number of short sequences to align against, you should place them in a single large file as individual fasta-format entries.

The squashed genome directory must contain only files produced by the SquashGenome program. Any other types of files will result in errors during the search. This prohibition extends to subdirectories as well.

## ELAND Alignments

Ensure the configuration file you use to run GERALD contains the following components:

- ▶ The path to your squashed genome files:  
`ELAND_GENOME /usr/local/share/eland/ncbi35`
- ▶ The path to your list of repeats (optional):  
`ELAND_REPEAT /usr/local/share/eland/refs30_5`  
This can significantly speed up alignment against large targets.
- ▶ The analysis variable to run ELAND:  
`ANALYSIS eland`
- ▶ Particular lanes that you want to analyze in the analysis variable:  
`34:ANALYSIS eland`  
This example indicates that lane 3 and 4 will be analyzed.



ELAND\_GENOME refers to a directory, not a file. The usual GERALD variables GENOME\_DIR and GENOME\_FILE are not used for ELAND analysis. ELAND expects a different file format other than fasta.

You can only specify one ELAND\_GENOME per lane.

After setting up the GERALD configuration file, you should be able to run “make.” The script `convertToFasta.pl` converts and concatenates all the reads into a single large fasta file which is then used as an input to ELAND. The script `convertFromELAND.pl` converts the results back into the by-tile PhageAlign format expected by the rest of the Pipeline.

ELAND operates on a lane-by-lane basis and uses up to 1 GB of memory. The Pipeline starts one ELAND job per lane. To prevent most computers from running out of memory, an artificial dependency in the GERALD makefile prevents multiple instances of ELAND from running at the same time. You can remove this limitation by using the following option in the GERALD configuration file:

```
ELAND_MULTIPLE_INSTANCES 8
```

Be aware that this may use up to 8 GB of memory on your analysis computer. If insufficient memory is available, the analysis is likely to crash. Allowed values for this option are 1, 2, 4, and 8. A value of 1 indicates no lane parallelization and uses up to 1 GB of RAM, a value of 2 indicates two parallel jobs and uses up to 2 GB, etc.

## Missing Bases in ELAND

Missing bases need to be specified as “N” characters and not “.” as in the sequence files. This conversion is managed automatically by GERALD but you need to be aware of it when running ELAND as a standalone program. For additional information on ELAND, see *Running ELAND as a Standalone Program* on page 68.

ELAND allows up to four external “N” characters at the beginning or end of the read. These bases are ignored.

ELAND allows up to two internal “N” characters, which are interpreted in one of two ways: “type D” for detection error and “type I” for insertion error.

- ▶ In a “type D” match, “N” indicates the base is there but not detected.

Read: ACNGT

Genome: ACCGT

- ▶ In a “type I” match, “N” indicates a base has been skipped.

Read: ACNGT

Genome: AC-GT

“Type D” and “type I” characters are given equal weight.

When lining up bases in your read with bases they align to in the reference, ignore any leading N characters and ignore any “type I” N characters because they are non-existent bases.

Most N characters are due to clusters wandering off the edge of the image for a cycle or two due to imperfect re-mapping of the tile position at different cycles. This produces a “type D” error. Otherwise, the Pipeline software will try to make a base call, even if the call is of low quality.

## Using ANALYSIS eland\_tag

For gene expression samples and other tag-counting applications, you can use ANALYSIS sequence to get purity-filtered sequences. These sequences are matched to the reference tag sets resulting in exact matches only. Using ANALYSIS eland\_tag to align experimental reads to a reference set produces not only exact matches but also one or two mismatches.

In addition to the standard output files, ANALYSIS eland\_tag creates a tag count file (s\_N\_tagcount.txt) for each lane that collapses and counts each distinct sequence. You can also specify GROUP\_LANES. For example, GROUP\_LANES 124 65 produces a file containing combined tag counts for each group of lanes in addition to a file for each lane.

ANALYSIS eland\_tag uses ELAND to align to a non-redundant set of annotation tags. Illumina provides human and mouse annotation that consists of a non-redundant set of all possible GATC+16 or CATG+17 sequences in the genome and transcriptome, choosing the best annotation for each distinct sequence. You may also use publicly available annotation for SAGE tags or generate your own.

### Squashing tag sets for eland

A separate fasta file should be prepared for each annotation tag set with one fasta header per file and each tag separated by Ns.

The following two examples show the beginning of a fasta file:

```
>mouseTranscrCanonicalCATG
AAAAAAAAAAAAAATCAC
NNNNNNNNNNNNNNNNNNNN
```

```

AAAAAAAAAAAACTTGA
NNNNNNNNNNNNNNNNN
AAAAAAAAAACAGCAA
NNNNNNNNNNNNNNNNN
AAAAAAAAACATAAAT
NNNNNNNNNNNNNNNNN
AAAAAAAAAGAAAAAA
NNNNNNNNNNNNNNNNN
AAAAAAAAATGGCTAA
NNNNNNNNNNNNNNNNN
AAAAAAAAATGGGTCA
NNNNNNNNNNNNNNNNN
AAAAAAAATCCTTATGT
NNNNNNNNNNNNNNNNN

```

```

>mouseMITO_CATG
CAAACCTCCATAGACCG
NNNNNNNNNNNNNNNNN
TGTAATTTTACCTCTAA
NNNNNNNNNNNNNNNNN
ACCAATGAACACTCTGA
NNNNNNNNNNNNNNNNN
AAATCTTCTGGGTGTAG
NNNNNNNNNNNNNNNNN
AACGGCTAAACGAGGGT
NNNNNNNNNNNNNNNNN
CTAGTCCCTAATTAAGG
NNNNNNNNNNNNNNNNN
AATATTTCAACAACAAA
NNNNNNNNNNNNNNNNN
TTCCTAGTTGTTTATAG
NNNNNNNNNNNNNNNNN
ACAAAAAATTGCTCCCC
NNNNNNNNNNNNNNNNN

```

The fasta files are squashed as any other reference genome files. The first four bases (CATG or GATC) are stripped from the annotation tags because they are part of the sequencing primer.

ANALYSIS eland\_tag aligns to one strand only. The annotation tags are non-symmetrical with a restriction site (GATC or CATG) on the left side only.

## Using ANALYSIS eland\_extended

ANALYSIS eland\_extended is an improved version of the ANALYSIS eland mode. ANALYSIS eland can align reads longer than 32 bases but demands that the first 32 bases of the read have a unique best match in the genome. The position of this match is used as a “seed” to extend the match along the full length of the read. ANALYSIS eland\_extended removes the uniqueness restriction by considering multiple 32 base matches to be considered and extended.



## Configuring ANALYSIS eland\_extended

There are two parameters that affect the output of the alignment, ELAND\_SEED\_LENGTH and ELAND\_MAX\_MATCHES. Both parameters can be specified lane-by-lane.

The following table describes the parameters for ANALYSIS eland\_pair.

**Table 11** Parameters for ANALYSIS eland\_extended

Parameter	Description
ELAND_SEED_LENGTH	By default, the first 32 bases of the read are used as a “seed” alignment. Setting ELAND_SEED_LENGTH to 25, will use 25 bases for the initial seed alignment. This should increase the sensitivity since two errors per 25 bases is less stringent than two errors per 32 bases. A read is more likely to be repetitive at the 25 base level than at the 32 base level, so a decrease in ELAND_SEED_LENGTH should probably be used in conjunction with an increase in ELAND_MAX_MATCHES. Setting this to very low values will drastically slow down the alignment time and will probably result in a lot of poor confidence alignments.
ELAND_MAX_MATCHES	By default, ANALYSIS eland_extended will consider at most 12 alignments of each read. ELAND_MAX_MATCHES allows the maximum number of alignments considered per read to be varied between 1 and 255.

Both ANALYSIS eland\_extended and ANALYSIS eland\_pair share a common export file that contains all read, quality value, and alignment information for a lane of data.

- ▶ ANALYSIS eland\_extended produces a single file per lane (s\_N\_export.txt).
- ▶ ANALYSIS eland\_pair produces two files, one for each of the two reads (s\_N\_1\_export.txt and s\_N\_2\_export.txt).

For a detailed description of the export.txt files, see *Text-Based Analysis Results* on page 58 and *Output File Formats* on page 86.

## Using ANALYSIS eland\_pair

Based heavily on ANALYSIS eland\_extended, ANALYSIS eland\_pair allows the analysis of a paired-read run using ELAND alignments. As part of the analysis, it will:

- ▶ Remap all clusters across both runs to the clusters found in the first cycle of the first read.
- ▶ Reset the matrix and matrix calculation after the end of the first read.
- ▶ Generate sequence strings whose combined length is equal to the sum of the lengths of each individual read.

The following files are produced for read 1 and read 2, and have identical format and function to the corresponding single-read files:

- ▶ s\_N\_1\_qraw.txt and s\_N\_2\_qraw.txt
- ▶ s\_N\_1\_eland\_query.txt and s\_N\_2\_eland\_query.txt
- ▶ s\_N\_1\_eland\_multi.txt and s\_N\_2\_eland\_multi.txt
- ▶ s\_N\_1\_frag.txt and s\_N\_2\_frag.txt
- ▶ s\_N\_1\_eland\_extended.txt and s\_N\_2\_eland\_extended.txt

The script `pickBestPair.pl` compares `s_N_1_eland_extended.txt` and `s_N_2_eland_extended.txt`, along with their quality values, and produces following two files of alignments, which contain pairing information:

- ▶ `s_N_1_saf.txt` and `s_N_2_saf.txt`

These files are used to do quality value recalibration and generate the following files, which contain calibrated quality values:

- ▶ `s_N_1_qcal.txt` and `s_N_2_qcal.txt`

The script `pickBestPair.pl` is then re-run using the calibrated quality values to obtain two more files of alignments:

- ▶ `s_N_1_calsaf.txt` and `s_N_2_calsaf.txt`

Finally, these files are parsed into the following output files:

- ▶ `s_N_1_export.txt` and `s_N_2_export.txt`
- ▶ `s_N_1_sorted.txt` and `s_N_2_sorted.txt`

Another output file produced is `s_N_anomaly.txt`, which contains reads that do not align. For some applications, reads that do not align may be of interest, since amongst those that are due to read errors may be some that represent genuine differences between the sequenced DNA and the reference.

For a detailed description of the `export.txt` files, see *Text-Based Analysis Results* on page 58 and *Output File Formats* on page 86.

## Configuring a Paired-Read Analysis

The alignments of the two reads that provide input to the pairing process may be varied by setting `ELAND_SEED_LENGTH` and `ELAND_MAX_MATCHES`. Both parameters may be set lane-by-lane, but the same values will apply to each of the two reads in a lane.

The paired-read analysis may be configured by passing options to `pickBestPair`. This is done by setting a parameter `PAIR_PARAMS` in the GERALD configuration file. For additional information, see *GERALD Configuration File* on page 35.

`PAIR_PARAMS` can be specified lane-by-lane. All of the options must be specified on a single line and space-separated, as in the following example:

```
8:PAIR_PARAMS --circular --min-percent-unique-pairs=30
```

The following table describes the parameters for ANALYSIS `eland_pair`.

**Table 12** Parameters for ANALYSIS `eland_pair`


Parameter	Description
<code>--circular</code>	This causes <code>pickBestPair</code> to treat each chromosome as circular and not linear, enabling it to detect valid pairings that “wrap around” when the two alignments are mapped onto the linear representation of the chromosome. [Optional] <code>--circular=my_mitochondria_file.fa</code> Treat alignments to <code>my_mitochondria_file.fa</code> as circular but other chromosomes as linear (as you might want to do when e.g. aligning to the whole human genome) <code>--circular=chromosome1:100000,chromosome2:300000</code> Specify chromosomes to circularize and specify the size to “wrap around” (possibly of use when the chromosome size is uncertain)

Table 12 Parameters for ANALYSIS eland\_pair (Continued)

Parameter	Description
--min-percent-unique-pairs	<p>A unique pair is defined as a read pair such that its constituent reads can each be aligned to a unique position in the genome without needing to make use of the fact that they are paired.</p> <p>pickBestPair works in a two-pass fashion:</p> <ol style="list-style-type: none"> <li>1. On the first pass it looks for all clusters that pass the quality filter and have a unique alignment of each of their two reads, then uses this information to determine the nominal insert size distribution and the relative orientation of the two reads.</li> <li>2. On a second pass this information is used to resolve repeats and other ambiguous cases.</li> </ol> <p>The number of unique pairs, expressed as a percentage of the total number of clusters passing filters, must exceed a certain percentage. Otherwise, no pairing is attempted and the two reads are effectively treated as two sets of single reads.</p> <ul style="list-style-type: none"> <li>• By default, this threshold is set to 30%.</li> <li>• For low quality data, a pairing can be forced by setting --min-percent-unique-pairs=5.</li> <li>• For some applications it may be useful to switch off the pairing completely. Set --min-percent-unique-pairs=101.</li> </ul>
--min-percent-consistent-pairs	<p>Of the unique pairs, the vast majority should have the same orientation with respect to each other. If they don't, it is indicative of the following problems:</p> <ul style="list-style-type: none"> <li>• Sample prep</li> <li>• Circularization is not switched on</li> <li>• A reference sequence is extremely diverged from the sample data</li> </ul> <p>In such cases, no pairing is attempted and the two reads are effectively treated as two sets of single reads.</p> <p>By default, the threshold for this parameter is set to 70%.</p>
--min-paired-read-alignment-score	<p>For each cluster, all possible pairings of alignments between the two reads are compared. This is the score of the best one. Since we are considering the two reads as one, both reads in a cluster get the same paired-read alignment score.</p> <p>The alignment score is nominally on a Phred scale. However, it is probably not safe to assume the calibration is perfect. Nevertheless, it is a good discriminator between good and bad alignments. The score must exceed this threshold to go in the sorted.txt file.</p> <p>The default value is zero.</p>
--min-single-read-alignment-score	<p>Each read is given a single-read alignment score.</p> <p>This is identical to the alignment score from an eland_extended analysis. If a read has a zero paired-read alignment score, but a single-read alignment score that exceeds this threshold, its alignment will still go in the sorted.txt files.</p> <p>If the alignments of the two reads can not be paired (resulting in a zero paired score) and only one of the reads has an alignment exceeding --min-single-read-alignment-score, the read pair is treated as a singleton. The alignment of the shadow read is unreliable enough to be ignored.</p> <p>The default value is zero.</p>

**Table 12** Parameters for ANALYSIS eland\_pair (Continued)

Parameter	Description
--add-shadow-to-singleton-threshold	If one read has a score exceeding --min-single-read-alignment-score but the other read either has no alignments or an alignment that does not exceed --min-single-read-alignment-score, then the non-aligning "shadow" read is added to the sorted.txt file with a zero alignment score, if the combined base quality of the shadow read (not alignment quality) exceeds this threshold. The default value of 1,000,000 indicates this feature is switched off.



## Chapter 5

# Analysis Output

### Topics

- 48 Introduction
- 48 Visual Analysis Summary
  - 48 Results Summary
  - 56 Cluster Intensity
  - 57 Error Rates
- 58 Text-Based Analysis Results
- 60 Interpretation of Run Quality
  - 60 Summary.htm
  - 64 IVC.htm
  - 64 All.htm and Error.htm

## Introduction

The Pipeline produces various text-based files and visual output during an analysis run. This section will help you interpret the various files that appear in an analysis output directory.

## Visual Analysis Summary

The results of an analysis are summarized as web pages that enable a large number of graphs to be viewed as thumbnail images. This section is intended to help you interpret the various graphs that appear in an analysis directory.

As the numbers of tiles and graphs have increased, it has become impractical to generate every possible graph for every tile. Therefore, the pages should be considered as a very basic view of the data.

### Results Summary

For each Run Folder, a Summary.htm file is produced, which contains comprehensive results and performance measures of your analysis run. It is located in the GERALD folder and provides an overview of quality metrics for a run with links to more detailed information in the form of pages of graphs. It is intended to load in a reasonable time; depending on the number of lanes and tiles used, the pages to which it links may take longer to display.



Although the Summary.htm file is an HTML file, it is also a valid XML file that is parseable by Perl's XML::Simple module. This means that you can mine the numbers in a Summary.htm file via a Perl script.

In the following descriptions of the tables included in Summary.htm, the terms chip and flow cell are used interchangeably.

### Chip Summary

The Chip Summary contains the instrument ID and the run folder. The Chip ID field is a placeholder that currently has a value of "unknown."

### Chip Results Summary

This table displays a summary of chip-wide performance statistics for the run. Both the original number of detected clusters and the number that passed quality filtering are shown. In addition, a chip yield in kilobases is presented. This is the sum over analyzed lanes of the product of number of quality-filtered clusters and number of bases per cluster used for analysis, excluding bases masked-out by a USE\_BASES directive.

## Lane Parameter Summary

Lane Parameter Summary records information about the sample in each flow cell lane and the analysis that has been specified for it.

- ▶ **Sample ID**—This is a placeholder field that currently has a value of “unknown.”
- ▶ **Sample Target**—The reference sequence against which reads from the sample in this lane are to be aligned. Depending on the analysis mode, this may be the name of a folder containing one or more sequence files or the name of an individual file. The acceptable file formats also depend on the analysis mode.
- ▶ **Sample Type**—Contains the analysis mode for reads from this lane.
- ▶ **Length**—The number of bases used per read (excluding any bases masked out using USE\_BASES). Where multiple reads are produced per cluster and a distinction is maintained between them during analysis, as in eland\_pair analysis of paired-end reads, their respective lengths will be listed.
- ▶ **Filter**—The criterion for clusters to be selected for analysis beyond the preliminary stages. Statistics for all detected clusters and for the subset that pass filtering are annotated as “raw” and “PF,” respectively, in Summary.htm.
- ▶ **Num Tiles**—The number of tiles from the lane that are used in the analysis.
- ▶ **Tiles**—A hyperlink for each lane to the location (within Summary.htm) of the statistics for individual tiles in that lane.

## Lane Results Summary

This table displays basic data quality metrics for each lane. Apart from Lane Yield, which is the total value for the lane, all the statistics are given as means and standard deviations over the tiles used in the lane

- ▶ **Clusters (raw)**—The number of clusters detected by the image analysis module of the Pipeline.
- ▶ **Clusters (PF)**—The number of detected clusters that meet the filtering criterion listed in Lane Parameter Summary.
- ▶ **1st Cycle Int (PF)**—The average of the four intensities (one per channel or base type) measured at the first cycle averaged over filtered clusters.
- ▶ **% intensity after 20 cycles (PF)**—The corresponding intensity statistic at cycle 20 as a percentage of that at the first cycle.
- ▶ **% PF Clusters**—The percentage of clusters passing filtering.
- ▶ **% Align (PF)**—The percentage of filtered reads that were uniquely aligned to the reference.
- ▶ **Alignment Score (PF)**—The average filtered read alignment score (reads with multiple or no alignments effectively contribute scores of 0).
- ▶ **% Error Rate (PF)**—The percentage of called bases in aligned reads that do not match the reference.

If `eland_pair` analysis has been specified for one or more lanes, then two Lane Results Summaries are produced, one for each read. All lanes for which analysis has been specified are represented in the Read 1 table, but only those for which `eland_pair` analysis has been specified contribute statistics to the Read 2 table.

## Expanded Lane Summary

This displays more detailed quality metrics for each lane. Apart from the phasing and prephasing information, all values are tile means for the lane.

- ▶ **Clusters (tile mean) (raw)**—The number of clusters detected by the image analysis module of the Pipeline.
- ▶ **% Phasing**—The estimated (or specified) value used by the Pipeline for the percentage of molecules in a cluster for which sequencing falls behind the current position (cycle) within a read.
- ▶ **% Prephasing**—The estimated (specification is not recommended) value used by the Pipeline for the percentage of molecules in a cluster for which sequencing jumps ahead of the current position (cycle) within a read.
- ▶ **% Error Rate (raw)**—The percentage of called bases in aligned reads from all detected clusters that do not match the reference.
- ▶ **Equiv Perfect Clusters (raw)**—The number of clusters in the ideal situation of read base perfectly predicting reference base that would provide the same information content (entropy of reference base given read base and a prior assumption of equiprobable reference bases) as calculated for all actual detected clusters.
- ▶ **% retained**—The percentage of clusters that passed filtering.
- ▶ **Cycle 2-4 Av Int (PF)**—The intensity averaged over cycles 2, 3, and 4 for clusters that passed filtering.
- ▶ **Cycle 2-10 Av % Loss (PF)**—The average percentage intensity drop per cycle over cycles 2–10 (derived from a best fit straight line for log intensity versus cycle number).
- ▶ **Cycle 10-20 Av % Loss (PF)**—The average percentage intensity drop per cycle over cycles 10–20 (derived from a best fit straight line for log intensity versus cycle number).
- ▶ **% Align (PF)**—The percentage of filtered reads that were uniquely aligned to the reference.
- ▶ **% Error Rate (PF)**—The percentage of called bases in aligned filtered reads that do not match the reference.
- ▶ **Equiv Perfect Clusters (PF)**—The number of clusters in the ideal situation of read base perfectly predicting reference base that would provide the same information content (entropy of reference base given read base and a prior assumption of equiprobable reference bases) as calculated for the actual clusters that passed filtering.

If `eland_pair` analysis has been specified for one or more lanes, then two Expanded Lane Results Summaries are produced, one for each read. All lanes for which analysis has been specified are represented in the Read 1 table, but only those for which `eland_pair` analysis has been specified contribute statistics to the Read 2 table.



## Per-Tile Statistics

Below the two types of lane summaries are per-tile statistics, grouped into a table for each lane. The statistics are a subset of those already presented in the Lane Results Summary, but are presented in these tables as averages over the detected (raw) or filtered (PF) clusters in individual tiles.

In the event that no clusters in a tile pass filtering, all the statistics for that tile are displayed within square brackets. Such an occurrence suggests an exceptional situation (e.g., a bubble) within the tile. The brackets indicate the tile has been excluded from the calculation of lane statistics and that the values are reported only for diagnostic purposes.

## Monotemplate Summary

This table appears after the per-tile summary table for lanes for which monotemplate analysis is specified. Statistics are presented for each monotemplate specified.

- ▶ **Lane**—Lane number.
- ▶ **Template**—The monotemplate sequence.
- ▶ **Count**—The number of reads that aligned to the monotemplate.
- ▶ **Percent**—The percentage of reads aligned to monotemplates that aligned to the current monotemplate.
- ▶ **True 1st Cycle Intensity**—The average intensity of the first base in reads aligned to the monotemplate.
- ▶ **Av Error Rate**—The average error rate over all cycles as a percentage of called bases for reads aligned to the monotemplate.
- ▶ **% Perfect**—The percentage of reads that are a perfect match to the monotemplate out of those that align to it.

## Pair Summary

For lanes for which eland\_pair analysis was performed, there are two per-tile summary tables (one for each read). These tables are preceded by a set of tables collectively entitled the Pair Summary. The Pair Summary tables provide statistics about the alignment outcomes of the two reads individually and as a pair, the latter including relative orientation and separation (insert size) of partner read alignments.

If the criteria for paired alignment are not met, the subset of tables reporting paired alignment results are replaced with the statement, "Paired alignment not performed."

The following tables are displayed in Pair Summary:

- ▶ Individual Alignments
- ▶ Unique Paired Alignments
- ▶ Unique Paired Alignment Effects
- ▶ Non-unique Paired Alignments
- ▶ Mispairing Rate
- ▶ Relative Orientation Statistics
- ▶ Insert Size Statistics
- ▶ Insert Statistics (% of individually uniquely alignable pairs)

**Individual Alignments**—This table displays the frequencies of the possible combinations of individual alignment outcomes within a pair. The following describe the possible outcomes:

- **Unique**—A unique alignment
- **Rescuable**—Multiple alignments such that a unique paired alignment could potentially be derived if the partner read is either unique or similarly rescuable
- **Repeat**—Multiple alignments such that consideration of the partner read will not help in selecting between them
- **Not Matched**—The read was not aligned
- **Low Quality**—The read contained too many uncalled bases to attempt alignment

The following example table indicates that 2,497,913 clusters from read 1 have a unique best alignment in the reference, and of those clusters, 2,443,861 from read 2 have a unique alignment. (In the example tables, the row labels pertain to read 1 and the column headings pertain to read 2.)

**Table 13** Example of Individual Alignments Table

Read 1 \ Read 2	Unique	Rescuable	Repeat	Not Matched	Low Quality	Total
Unique	2443861 (93.2%)	17547 (0.7%)	395 (0.0%)	31818 (1.2%)	4292 (0.2%)	2497913 (95.3%)
Rescuable	17670 (0.7%)	54356 (2.1%)	929 (0.0%)	897 (0.0%)	118 (0.0%)	73970 (2.8%)
Repeat	405 (0.0%)	920 (0.0%)	289 (0.0%)	9 (0.0%)	2 (0.0%)	1625 (0.1%)
Not Matched	26724 (1.0%)	776 (0.0%)	10 (0.0%)	6928 (0.3%)	240 (0.0%)	34678 (1.3%)
Low Quality	138 (0.0%)	2 (0.0%)	0 (0.0%)	19 (0.0%)	14037 (0.5%)	14196 (0.5%)
Total	2488798 (94.9%)	73601 (2.8%)	1623 (0.1%)	39671 (1.5%)	18689 (0.7%)	2622382 (100.0%)

ELAND maintains a list of best alignments for each read. By default, this list can contain at most 10 alignments. The maximum size of the list may be increased (or decreased) by specifying `ELAND_MAX_MATCHES` in the GERALD config file. This can be specified on a lane-by-lane basis.

Reads for which ELAND has retained a list of alignments may potentially be “rescued” if only one of the possible alignments has the correct insert size and orientation with respect to the other read of the cluster. These are called “rescuable.”

Not all rescuable reads are rescued. This is dependent on the read pairing algorithm finding a single consistent pairing of the reads among the list of possibilities. The ability of the read pairing algorithm to do this depends on the quality of the sample prep and the degree of structural variation between the reference sequence being aligned to and the sample being sequenced. Conversely, if the number of possible alignments exceeds `ELAND_MAX_MATCHES`, then ELAND stores only the number of matches found. There is no potential for picking the best alignment, so the read is classified as a “repeat.”

If the analysis were re-run with a higher value of `ELAND_MAX_MATCHES`, some reads that were classified as repeats would become rescuable, but the sum of rescuable and repeat reads will stay the same.

**Unique Paired Alignments**—This table breaks down the unique paired alignments according to the alignment outcomes of their component reads, which can only be “unique ”or “rescuable.”

If at least one alignment exists for each of the two reads in a cluster, then there are three possible outcomes:

1. **Unique Paired Alignment**—It is only possible to pick one alignment for read 1 and one alignment for read 2 such that the relative orientation of the two alignments and the distance between the alignment positions are consistent with the sample.
2. **Non-Unique Paired Alignment**—There is more than one way to pick one alignment for read 1 and one alignment for read 2 such that the relative orientation of the two alignments and the distance between the alignment positions are consistent with the sample.
3. **Inconsistent Pair**—There is no way to pick one alignment for read 1 and one alignment for read 2 such that the relative orientation of the two alignments and the distance between the alignment position are both consistent with the sample.

The following example indicates that 2,474,278 clusters have one possible alignment of each read such that the two reads have the appropriate relative position and orientation with respect to one another. Of these clusters, 2,437,627 have only one possible alignment of each of the two reads. In other cases, the read pairing algorithm has to pick from a list of multiple possibilities for one or both of the two reads.

**Table 14** Example of Unique Paired Alignments Table

Read 1 \ Read 2	Unique	Rescuable	Total
Unique	2437627 (98.5%)	17125 (0.7%)	2454752 (99.2%)
Rescuable	17266 (0.7%)	2260 (0.1%)	19526 (0.8%)
Total	2454893 (99.2%)	19385 (0.8%)	2474278 (100.0%)

The frequencies in the Unique Paired Alignments table are often somewhat lower than the corresponding values in the Individual Alignments table due to the conditions required for a unique paired alignment. For example, even when both reads are individually uniquely aligned, it is possible that their relative positions or orientations are not compatible with the paired read model.

Some reads are reflected in the Unique Paired Alignments table. Alignment positions that are unexpectedly far apart, unexpectedly close together, or even on different chromosomes, could be due to an error or represent a genuine feature in the sample (deletion, insertion, or translocation).

**Unique Paired Alignment Effects**—The Unique Paired Alignment Effects table is similar to the Unique Paired Alignments table, but focuses upon what proportions of unique paired alignments were rescued (one or both of the partner reads were not individually uniquely aligned).

The example Individual Alignments table (page 52) shows that 17,547 reads were such that read 1 was unique and read 2 was potentially rescuable. The following table indicates that unique alignments were found for 17,125 (97.6%) of these reads.

**Table 15** Example of Unique Paired Alignment Effects Table

Effect \ Read	Read 1	Read 2	All Reads	Total
Uniqueness Rescue (% of rescuable)	17266 (97.7%)	17125 (97.6%)	2260 (4.2%)	36651 (40.9%)

**Non-unique Paired Alignments**—This table breaks down the non-unique paired alignments according to the alignment outcomes of their component reads. Paired alignment is attempted only for pairs where the individual alignments of both partners are either unique or rescuable.

The example table below shows that a small number of clusters are non-unique even though there is a unique alignment of one of the two reads. This means there is more than one alignment of the other read that is consistent with alignment of the first, which might occur if it aligns to a tandem repeat or low-complexity region.

**Table 16** Example of Non-unique Paired Alignments Table

Read 1 \ Read 2	Unique	Rescuable	Total
Unique	0 (0.0%)	58 (0.1%)	58 (0.1%)
Rescuable	63 (0.1%)	51957 (99.8%)	52020 (99.9%)
Total	63 (0.1%)	52015 (99.9%)	52078 (100.0%)

**Mispairing Rate**—Mispairing is considered to happen when one read of a pair can be aligned (whether the alignment is unique, rescuable, or against a repeat) and the other read can not be aligned because it is of low quality or simply no match can be found for it in the reference.

For example, in the Individual Alignments table (page 52), you will see that of the clusters with a read 1 that could not be aligned, 26,724 had a read 2 that uniquely aligned, 776 had a read 2 that was rescuable, and 10 had a read 2 that was a repeat. Of the clusters with a low-quality read 1, 138 had a read 2 that uniquely aligned, 2 had a read 2 that was rescuable, and none had a read 2 that was a repeat. The sum of these numbers is the value of Read 1 Lost in the Mispairing Rate table.

**Table 17** Example of Mispairing Rate Table

Read 1 Lost	Read 2 Lost
27650 (1.1%)	37136 (1.4%)

**Relative Orientation Statistics**—The relative orientation of a pair is the orientation of read 2 relative to the orientation of read 1, based on the definition that the read 1 orientation is forward. The relative orientation is defined as positive if the read 2 position is greater than the read 1 position.

These statistics are given only for those pairs in which both reads were individually uniquely aligned, since these are the reads used to determine the predominant relative orientation. Other orientations are considered anomalous and are filtered out.

The symbols used in the column headings are intended as a visual reminder of the definitions of the four possible relative orientations. In the example below, the nominal orientation is correctly computed as the two reads “pointing to” each other, as expected for the standard Illumina short insert paired-read sample prep.

**Table 18** Example of Relative Orientation Statistics Table

F-: > R2 R1 >	F+: > R1 R2 >	R-: < R2 R1 >	R+: > R1 R2 <	Total
184 (0.0%)	161 (0.0%)	243 (0.0%)	2443273 (100.0%)	2443861

**Insert Size Statistics**—Statistics are derived from the insert sizes of those pairs in which both reads were individually uniquely aligned and have the predominant relative orientation. First, the median is determined. Then, a standard deviation value is determined independently for those values below the median and those above it. The lower and upper thresholds for acceptable insert sizes are then defined as three of the relevant standard deviations below and above the median, respectively.

**Table 19** Example of Insert Size Statistics Table

Median	Below-median SD	Above-median SD	Low Thresh.	High Thresh.
214	10	11	184	247

**Insert Statistics (% of individually uniquely alignable pairs)**—This table shows the number of inserts (out of those used to calculate insert size statistics) considered acceptable in size and of those falling outside the thresholds displayed in the Insert Size Statistics table. The percentages are relative to the original number of pairs in which both reads were individually uniquely aligned.

For example, the Individual Alignments table (page 52) showed that 2,443,861 clusters were such that read 1 and read 2 each had a unique alignment. The Unique Paired Alignment table (page 53) showed that 2,437,627 clusters have a unique paired alignment. This leaves 6,234 clusters to account for. Together, the Relative Orientation Statistics table (page 55) and the following Insert Statistics table describe the outcome for these clusters:

- 3,945 had a correct orientation but had an implied insert size that was too small.
- 1,701 clusters were correctly oriented but had an implied insert size that was too large.

- 184, 161, and 243 clusters were oriented in the three possible ways an orientation can be wrong.

These five figures total 6,234, as expected.

**Table 20** Example of Insert Statistics Table

Too Small	Too Large	Orientation and Size OK
3945 (0.2%)	1701 (0.1%)	2437627 (99.7%)

## Cluster Intensity

Key web pages that illustrate cluster intensity are IVC.htm and All.htm.

### IVC.htm

This file contains plots that display lane averages over all tiles in the lane. The plots displayed are All, Called, %Base\_Calls, %All, and %Called.

- ▶ **All**—This is the lane average of the data displayed in All.htm. It plots each channel (A, C, G, T) separately as a different colored line. Means are calculated over all clusters, regardless of base calling. If all clusters are T, then channels A, C, and G will be zero. If all bases are present in the sample at 25% of total and a well-balanced matrix is used for analysis, the graph will display all channels with similar intensities. If intensities are not similar, the results could indicate either poor cross-talk correction or poor absolute intensity balance between each channel.
- ▶ **Called**—This plot is similar to All, except means are calculated for each channel using clusters that the base caller has called in that channel. If all bases are present in the sample at 25% with pure signal (zero intensity in the non-called channels), the Called intensity will be four times that of All, as the intensities will only be averaged over 25% of the clusters. For impure clusters, the difference in intensity will be less than four times that of All.

The Called intensities are independent of base representation, so a well-balanced matrix will display all channels with similar intensities.

- ▶ **%Base\_Calls**—The percentage of each base called as a function of cycle. Ideally, this should be constant for a genomic sample, reflecting the base representation of the sample. In practice, later cycles often show some bases more than others. As the signal decays, some bases may start to fall into the noise while other still rise above it. Matrix adjustments may help to optimize data.
- ▶ **%All and %Called**—Exactly the same as All and Called, but expressed as a percentage of the total intensities. These plots make it easier to see changes in relative intensities between channels as a function of cycle by removing any intensity decay.

For information on interpreting results in the IVC.htm file, see *Interpretation of Run Quality* on page 60.

## All.htm

This file gives a tile-by-tile representation of the mean matrix-adjusted intensity of clusters plotted as a function of cycle. It plots each channel (A, C, G, T) separately as a different colored line. Means are calculated over all clusters, regardless of base calling.

If all clusters are T, channels A, C, and G will be at zero. If all bases are present in the sample at a rate of 25% and a well-balanced matrix is used for analysis, the graph will display all channels with similar intensities. If intensities are not similar, the results could indicate either poor cross-talk correction or poor absolute intensity balance among each channel.

A genome rich in GC content may not provide a balanced matrix for accurate cross-talk correction and absolute intensity balance.



For large experiments (> 200 tiles per lane), All.htm, Perfect.htm, and Error.htm only show a subset of tiles. However, each file contains links to the full output results. For example, Error.htm links to FullError.htm. The full output files may take some time to open.

## Error Rates

For all analysis modes except sequence, Perfect.htm and Error.htm are produced, which measure sequence error rates.

### Perfect.htm

This graph shows the proportion of reads in a tile that have 0, 1, 2, 3, or 4 errors by the time they get to a given cycle.

Good data show a high proportion of reads with zero errors throughout the cycles.

### Error.htm

This file shows a graph of error rates for each tile on a flow cell. The red bar shows the percentage of bases at each cycle that are wrong, as calculated based on alignment to the reference sequence. Issues such as focus or fluidics problems manifest themselves as spikes in the graph.

Good data is 1–1.5% or less for 25 aligned bases.

- ▶ PhageAlign allows any number of errors in an alignment and provides an accurate count of the error rate. However, it is too slow for aligning against target references larger than 2 Mb.
- ▶ ELAND is capable of aligning against large genomes, such as human, in reasonable time. However, it allows only two errors per fragment. This means that error rates based on ELAND alignments are underestimated. Very poor quality data has more than two errors in the first 32 aligned bases and is excluded from the calculations.

## Text-Based Analysis Results

The output files for each lane of a flow cell are named using the format `s_N_sequence.txt`, where N represents a specific lane of the flow cell. For paired-read analysis, there are two parallel output files, one for each read. The files are named using the format `s_N_R_sequence.txt`, where N represents a specific lane of the flow cell and R represents the read number. The files are found in the GERALD folder of a finished analysis run.

The output files for each tile are named using the format `s_N_TTTT_realign.txt`. For example, all files pertaining to tile 23 of lane 3 have names starting with `s_3_0023`.

The following table lists the files that contain the most meaningful data produced from your analysis run and the GERALD analysis mode that creates them. For descriptions of the GERALD analysis variables, see *ANALYSIS Variables* on page 29.

**Table 21** Text-Based Analysis Results

GERALD Analysis Mode	Output File	Description
All modes except ANALYSIS none	<code>s_N_sequence.txt</code>	This file contains all sequences in a single lane of a flow cell in an exportable format. The content of this file is affected by the following parameters: USE_BASES, QF_PARAMS, SEQUENCE_FORMAT, QUALITY_FORMAT. For a description of each of these parameters, see <i>ANALYSIS Variables</i> on page 29.
ANALYSIS default ANALYSIS eland	<code>s_N_realign.txt</code>	This file contains filtered alignment information.
	<code>s_N_rescore.txt</code>	This file contains error rates for filtered data based on the alignments in the <code>rescore.txt</code> files. These are used to create the graphs in the <code>Error.htm</code> pages.
	<code>s_N_qreport.txt</code>	This file reports the accuracy of the base calling quality values, making use of the <code>_qraw.txt</code> files.
	<code>s_N_qcalreport.txt</code>	This file reports the accuracy of the recalibrated quality values, making use of the <code>_qcal.txt</code> files if they are present for the type of analysis you have specified. The format is identical to <code>s_N_qreport.txt</code> .
ANALYSIS eland_extended	<code>s_N_export.txt</code>	This file contains the results of alignment of all reads in the lane. The fields are tab separated to facilitate export to databases. This file has a line for <b>every</b> read, not just those that pass purity filtering. The last field on each line is a flag telling you whether or not the read passed the filter (1 or 0). For file formats, see <i>Output File Formats</i> on page 86.
	<code>s_N_sorted.txt</code>	This output file is similar to <code>s_N_export.txt</code> , except it contains only entries for reads which pass purity filtering <b>and</b> have a unique alignment in the reference. These are sorted by order of their alignment position, which is meant to facilitate the extraction of ranges of reads for purposes of visualization or SNP calling.



Table 21 Text-Based Analysis Results (Continued)

GERALD Analysis Mode	Output File	Description
ANALYSIS eland_pair	s_N_1_sequence.txt, s_N_2_sequence.txt	These parallel sets of files contain filtered sequences for each lane.
	s_N_1_export.txt, s_N_2_export.txt	These parallel sets of files contain the results of alignment of all reads in the lane. The fields are tab separated to facilitate export to databases. Each file has a line for <b>every</b> read, not just those that pass purity filtering. The last field on each line is a flag telling you whether or not the read passed the filter (1 or 0). For information on file format, see <i>Output File Formats</i> on page 86.
	s_N_1_sorted.txt, s_N_2_sorted.txt	These parallel sets of files are similar to s_N_1_export.txt and sN_2_export.txt, except they contain only entries for reads which pass purity filtering <b>and</b> have a unique alignment in the reference. These are sorted by order of their alignment position, which is meant to facilitate the extraction of ranges of reads for purposes of visualization or SNP calling.
	s_N_anomaly.txt	This file contains one line for each read for which the two halves of the read did not align with a nominal distance and orientation from each other. This is the file to mine for structural variation information.

For descriptions of file formats, see *Output File Formats* on page 86.

Numerous intermediate files are produced during an analysis run. For a description of these files, see *Intermediate Output Data Files* on page 83.

## Interpretation of Run Quality

After the analysis of a run is complete, you need to interpret the data in the report summary and various graphical outputs. This section describes a standard, systematic way to examine your data.

The starting point is to know what a standard run of acceptable quality looks like. This is something of a moving target and is dependent on individual instruments, instrument configuration, genomic sample type, type of analysis, flow cell preparation, and the current state of the art. Therefore, the numbers shown in this section are for example only.

### Summary.htm

The Summary.htm file is the first file you should review after your analysis is complete.

The following are examples of two of the tables found in Summary.htm, Lane Results Summary and Expanded Lane Summary, each truncated to a single lane of information. For a description of the tables found in Summary.htm, see *Results Summary* on page 48.

**Table 22** Example of Lane Results Summary

Lane	Clusters	Av 1st Cycle Int	% intensity after 20 cycles	% PF Clusters	% Align (PF)	AV Alignment Score (PF)	% Error Rate (PF)
1	23621 ± 407	1926 ± 60	65.12 ± 2.48	52.55 ± 0.37	98.33 ± 0.14	2855.55 ± 90.70	6.71 ± 0.63

**Table 23** Example of Expanded Lane Summary

Lane Info		Phasing Info		Raw Data		Filtered Data						
Lane	Clusters	% Phasing	% Pre-phasing	% Error Rate (Raw)	Equiv Perfect Clusters (raw)	% Re-tained	Cycle 2-4 Av Int	Cycle 2-10 Av % Loss	Cycle 10-20 Av % Loss	% Align (PF)	% Error Rate (PF)	Equiv Perfect Clusters (PF)
1	23621	0.9300	0.5800	11.17	12457	52.55	1728 ± 43	2.31 ± 0.24	181 ± 0.15	98.33	6.71	9709

The key parameters that you should examine are listed in the following sections, along with conditions, possible causes for those conditions, and suggested actions to correct the condition.

## Clusters

This column contains the average number of clusters per tile detected in the first cycle images. For 1 Gbases of data at 35 cycles, this value needs to be greater than 20,000.

Condition	Possible Cause	Suggested Action
<b>Fewer clusters than expected:</b>		Reanalyze with new default offsets. If the problem persists, ensure that the alignment config file contains "SIMILARITY" filtering. The use of "SIMILARITY" filtering will result in low numbers passing filters.
Few bright clusters on the flow cell	Problem with cluster formation	
Blurred images	Poor focus or dirty flow cell surface	
Lots of clusters visible	Cluster density or size is too great to distinguish individual objects	
<b>More clusters than expected:</b>		
Too many clusters on the flow cell	Problem with cluster formation	
Very large clusters	Double counting	

## Average First Cycle Intensity

Generally, brighter is better, but this result is instrument and sample dependent. Ideally, this value should be greater than 1000. For some paired-end sample preparations, this value should be greater than 500.

Condition	Possible Cause
Low intensity	Problem with cluster formation or poor focus

## Percentage of First Cycle Intensity Remaining After 20 Cycles of Sequencing

Generally, the higher, the better. Greater than 50% is acceptable, though it can be sample dependent.

Condition	Possible Cause	Suggested Action
Low value	A correct measure of rapid signal decay deduced from intensity plots	Check experiment fluidics or temperature control
	Problem with cycle 20 deduced from intensity plots.	Check fluidics and focus for this cycle
Exceptionally high value	Low first cycle intensity	Check first cycle focus

## Percentage of Clusters Passing Filters

To remove the least reliable data from the analysis, the raw data can be filtered to remove any clusters that have “too much” intensity corresponding to bases other than the called base. By default, the purity of the signal from each cluster is examined over the first 12 cycles and  $CHASTITY = \frac{\text{Highest\_Intensity}}{\text{Highest\_Intensity} + \text{Next\_Highest\_Intensity}}$  is calculated for each cycle. If  $CHASTITY > 0.6$  for all 12 cycles, then the cluster passes the filters. Both  $CHASTITY > 0.6$  and 12 cycles are essentially arbitrary, and are a compromise arrived at to remove most of the error prone data without throwing away too much of the good data.

The higher the value, the better. Ideally, a value of  $> 70\%$  good, but this value is very dependent on cluster density. When there is above 20,000 clusters per tile, the percentage starts fall, since the major cause of an impure signal in the early cycles is the presence of another cluster within a few micrometers.

Condition	Possible Cause	Suggested Action
Very few clusters passing filter	<ul style="list-style-type: none"> <li>Poor flow cell, perhaps unblocked DNA</li> <li>Faint clusters</li> <li>Out of focus</li> <li>Poor matrix</li> <li>A fluidics or sequencing failure</li> <li>Bubbles in individual tiles</li> <li>Too many clusters</li> <li>Large clusters</li> <li>High phasing or prephasing</li> </ul>	<p>Some of the causes may be at a single cycle. If the problem is isolated to these early cycles, it is possible that this filtering throws away very good data.</p> <p>Base calling errors may be limited to effected cycles, and as early cycles are fairly resistant to minor focus and fluidics problems, even the number of errors may be few. The filtering can always be set manually to some other values. Check before assuming all the data are poor.</p>

## Percentage of Clusters Passing Filters that Align Uniquely to the Reference Genome

Optimal value depends on the genome sequenced and the read-length; the higher (up to 100% max), the better. For example, for 30-mers and human genome, the optimum is less than 80%.

This result is genome specific and dependent on the completeness of the reference. A failure to align could be due to repeat or missing regions, or due to indels where sample and reference do not match.

Condition	Possible Cause	Suggested Action
Much lower than expected when using ELAND	Fluidics or instrument problem	Look for an intensity dip in IVC plots. If there is a problem and it occurs after a sufficiently useful read-length, re-run ELAND analysis using only the “good” cycles before the instrument problem.
	Contamination from other genetic material resulting in an inability to align data	Align a few sample tiles with PhageAlign. Genomic contamination will show as early cycle error rates. If error rates remain fairly constant with cycle, then the “correct” genome has probably sequenced correctly. Non smooth error rate plots or IVC plots indicate the presence of specific tags or sequences.

## Percentage Error Rate of Clusters Passing Filters

This value should be as low as possible, but it is very dependent on read-length. At 32 cycles, the error rate should be around 2%. Depending on the quality of the data, it will tend to rise at this point. If there is a sudden rise beyond cycle 32, then it is likely that ELAND has effectively filtered out many clusters with more than two errors, thus suppressing the true error rate up to this point. The percentage aligning will also be low.

With PhageAlign analysis of control samples, error rates for 25 cycles should be < 1.5%.

## Percentage of Phasing and Prephasing

Ideally, these values should be as low as possible. Satisfactory results can be obtained with up to 1% for each. Preferably, they should be closer to 0.5%.

Condition	Possible Cause	Suggested Action
High phasing or prephasing	Reagent issue (reagents have deteriorated) Fluidics	Check for leaks or bubbles in images or early cycle discrepancies in intensity plots.
	Poor flow cell	Poor blocking can be evident as intensity in all channels from cycle 1.

## Standard Deviations

Many values have standard deviations associated with them. This can be the first indication as to the uniformity of the flow cell. If standard deviations are high, then it indicates variability from tile to tile with a lane.

Condition	Possible Cause	Suggested Action
High standard deviations	Check poor tiles for: <ul style="list-style-type: none"> <li>• Bubbles</li> <li>• Focus</li> <li>• Dirty flow cell surface</li> </ul>	Look at the tile-by-tile statistics that appear below the flow cell-wide summary.

After reviewing the tables in Summary.htm, examine the thumbnails, and the output files IVC.htm, All.htm, and Error.htm.

**IVC.htm**

For a detailed description of the plots found in the IVC.htm file, see *IVC.htm* on page 56.

Condition	Possible Cause
Intensity curves are not smooth	Cycle to cycle focus or fluidics problems
Called intensities are not equal ("% Called" may be +/- 5% out without major problems)	Poor fluidics or poorly blocked flow cell If from cycle 1, initial matrix estimate may also be in error

**All.htm and  
Error.htm**

The results in both files should show consistency from tile to tile down a lane and from lane to lane, if the results are from the same sample.

Condition	Possible Cause
Tile variability	Bubbles Rapid focus fluctuations Dirty flow cell surface
Rising error rates (Rates will always rise eventually at high read-lengths)	Low intensity at start High decay rate High phasing or prephasing
High, but constant error rates from cycle 1	Genomic contamination



## Chapter 6

# Advanced Pipeline Usage

### Topics

- 66 Introduction
- 66 Running Bustard as a Standalone Program
  - 66 Assigning a Control Lane
- 67 Running GERALD as a Standalone Program
  - 67 Additional "Make" Options
- 68 Running ELAND as a Standalone Program
  - 69 Compiling ELAND
  - 69 Command Line Syntax

## Introduction

Bustard, GERALD, and ELAND may be run as standalone programs. This allows you to rerun your analysis using different parameter settings without running the rest of the Pipeline. You can rerun the base caller on a different subset of intensity files, perform alignments on the same base-called sequences, or rerun sequences against another genome.

## Running Bustard as a Standalone Program

You can invoke the base calling script repeatedly without rerunning the image analysis. This lets you run the base caller on a different subset of cycles, tiles, and lanes with different parameter settings. The run is set up by a separate script called `bustard.py`.

The following example shows the various options you can specify with the base calling script:

```
/path/Pipeline/Goat/bustard.py
  [--cycles=1-25|auto] [--tiles=s_1,s_2_0003,...]
  [--matrix=mymatrix.txt|auto|auto<n>]
  [--phasing=0.01|auto|auto<n>] [--prephasing=0.01]
  [--make]
  [--GERALD=/path/config.txt] [--control-lane=<lane>]
  <Firecrest directory>
```

For example, the following command calls the base calling script and points to the image analysis directory:

```
/path/Pipeline/Goat/bustard.py
  /data/070813_ILMN-1_0217_FC1234/Data/C1-
  27_Firecrest1.9.0_23-08-2007-user
```

This will not generate any makefiles and directories unless the “make” option has been specified.

### Assigning a Control Lane

If you need to assign a control lane for more accurate matrix and phasing estimation, run base calling using the `bustard.py` script and use `control-lane` as an argument.

```
Pipeline/Goat/bustard.py --control-lane=4 --make /
  data/070813_ILMN-1_0217_FC1234/Data/C1-
  26_Firecrest*
```

Change to the newly generated Bustard folder and type the “make all” command.

```
make all
```



## Running GERALD as a Standalone Program

You can run an analysis using GERALD without the rest of the Pipeline if you want to perform alignments with different parameters on the same base-called sequences.

GERALD uses a text-based configuration file containing all parameters required for alignment, visualization, and filtering. These parameters are the type of analysis to perform, which bases to used for alignment, and the reference files for a sequence alignment. The GERALD.pl script is used to generate the GERALD makefile. The makefile is executed using the “make” utility.

A typical invocation would be as follows:

```
Pipeline/Gerald/GERALD.pl gerald_config.txt
--EXPT_DIR path_to_bustard_folder --FORCE
```

The standard way to run GERALD is to set the parameters in a configuration file, create a makefile, and start the analysis with the “make” command.

1. Edit the config.txt file as described in *GERALD Configuration File* on page 35.
2. Enter the following command to create a makefile for sequence alignment. To generate a makefile in GERALD, use FORCE instead of “make.”

```
GERALD.pl config.txt --FORCE
```

3. Change to the newly created GERALD folder under the “path\_to\_bustard\_folder.” Type the “make” command for basic analysis.

You may prefer to use the parallelization option as follows:

```
make -j 3 all
```

The extent of the parallelization depends on the setup of your computer or computing cluster. For a description of parallelization, see *Using Parallelization* on page 93.

For more information on GERALD, see *Using GERALD* on page 27.

### Additional “Make” Options

You may perform a partial build of your analysis. This feature may be useful for a sneak preview of your results, after which a full analysis may be built as described above. For example, to build all files for tile 12 of lane 3, use the following “make” option:

```
make TILE=s_3_0012
```

You may specify specific tiles to perform a partial build of your analysis. The following example will build tiles 0005, 0010, 0015, 0020, and 0025 from lanes 3 and 6:

```
make TILE=s_[36]_00[0-2][05]
```

This example specifies any tile for which the last digit is 0 or 5, the previous digit is 0, 1, or 2, and the previous two positions are 00.

## Running ELAND as a Standalone Program

You can run ELAND without the rest of the Pipeline as a post-analysis step. ELAND can be run as a standalone program for the following reasons:

- ▶ To test the effect of different filter parameters
- ▶ To test alignment targets
- ▶ To test applications that read export files

The `eland_extended` and `eland_pair` analysis modes introduced in Pipeline version 0.3 share a common export file format. The intention of this file is to combine all the important information for a lane into one file (or two files in the case of a paired-read run) with the following results:

- ▶ Bring in the base quality value information.
- ▶ Use the base quality values to pick the best alignment.
- ▶ Give a quality score to the alignment (generated by running several scripts to post-process the raw ELAND output).

To run ELAND as a standalone program, use the script `Pipeline/Eland/ELAND_standalone.pl`.

```
./Pipeline/Eland/ELAND_standalone.pl -if read1.fastq -
  if read2.fastq -it fastq
-eg /lustre/data01/Mondas_software/Genomes/
  E_coli_ELAND
```

**Table 24** Options for `ELAND_standalone.pl`

Option	Short Form	Description
<code>--input-file</code>	<code>-if</code>	Specify at least one file for single-reads and two files for paired-reads
<code>--input-type</code>	<code>-it</code>	Type of input file (fastq, fasta, or export)
<code>--read-length</code>	<code>-rl</code>	This value will be determined from the input type, if not specified
<code>--seed-length</code>	<code>-sl</code>	Length of read substring (seed) used for ELAND alignment
<code>--eland-genome</code>	<code>-eg</code>	Full path of a squashed genome directory
<code>--output-prefix</code>	<code>-op</code>	Produces a set of output files with a prefix of this value (default value is "reanalysis")
<code>--pipeline-dir</code>	<code>-pd</code>	Path of the Pipeline installation (by default it is the same directory where the executable is found)
<code>--pair-params</code>	<code>-pp</code>	Indicates paired-read analysis parameters to pass to <code>pickBestPair</code> . Multiple arguments must be contained in quotation marks. Defaults to <code>--circular</code> (treats all chromosomes as circular)



### NOTE

For paired-read analysis, both reads must share the same `input-type`, `read-length`, and `seed-length`.

Running ELAND as a standalone program does not perform all of the various steps that are included during a GERALD run. For example:

- ▶ Quality value recalibration
- ▶ Extension of alignments beyond 32 bases
- ▶ Removal of sequences that fail signal purity filtering

If you require any or all of the above, it is best to create a modified config file to align to a different squashed genome, and rerun GERALD. For more information, see *GERALD Configuration File* on page 35.

## Compiling ELAND

ELAND is compiled automatically as part of the Pipeline installation as described in *Installation Prerequisites* on page 75.

You can manually compile ELAND from the Pipeline/Eland directory using the “make” command. This compiles ELAND without compiling the rest of the Pipeline.

```
make -e eland
```

## Command Line Syntax

Use the following command line syntax to run ELAND as a standalone program:

```
eland_executable queryFile squashedGenomeDir
  [output_file].txt
  [--multi[=N0[,N1,N2]] [repeatFile]
```

### queryFile.txt

queryFile.txt is a file of query sequences. This must be either a multi-entry fasta format file or a one-sequence-per-line ASCII file. The length of each sequence must exceed the read length specified at compilation. Unspecified bases in the reads must be denoted by an “N.” IUPAC ambiguity codes are not handled.

### squashedGenomeDir

squashedGenomeDir is the path to the directory of squashed genome files. For more information, see *Preparing the Reference Genome* on page 38.

### [output\_file].txt

The ELAND output file contains the initial output of the ELAND alignment program with one line of output per sequence. The name of the output file depends on the analysis you are running.

- ▶ ANALYSIS eland produces an output file named s\_N\_eland\_results.txt.
- ▶ ANALYSIS eland\_extended and ANALYSIS eland\_pair are run with the --multi option, and produce an output file named s\_N\_eland\_multi.txt.

For an explanation of intermediate output files, see *Intermediate Output Data Files* on page 83.

For a description of the output file format, see Table 29 on page 87.

## **--multi**

If `--multi` is specified, ELAND will store and display multiple (10 by default) matches for each read.

- ▶ `--multi=20,40,80` will display at most 20 exact matches, 40 single-error matches, and 80 2-error matches.
- ▶ `--multi=20` will display at most 20 matches of any number of errors.

For a description of output file formats using the multi option, see Table 29 on page 87.

## **repeatFile.txt**

You may want to specify a set of words that you know are highly repetitive in your target files at your read length of interest. You can then tell ELAND to ignore them, which greatly increases the speed of whole-human-genome alignments. There is no automatic way of generating a repeat file, but with a bit of Perl/shell scripting, it is straightforward to extract a list of repeats from the output of a few ELAND runs to improve the speed of future runs.

You can run the basic test harness script `ELAND_test.pl` from the ELAND directory to verify correct operation.



## Appendix A

# System Requirements and Software Installation

### Topics

72	Introduction
72	System Requirements
72	Network Infrastructure
73	Analysis Computer
75	Installation Prerequisites
75	Setting Up Email Reporting
77	Installing the Pipeline Software
77	Compiling on Other Platforms
77	Directory Setup

## Introduction

This section describes the Pipeline system requirements and the software installation instructions. It also describes how to set up your instrument directory.

## System Requirements

Images are acquired and stored on the Genome Analyzer. They must be transferred to an external computer to be analyzed by the analysis software, which handles image processing, base calling, and sequence alignment. Based on an eight-lane flow cell with three columns and 110 rows per lane, each sequencing run generates approximately 1 TB of data during a full 2–3 day run. Paired-end runs generate approximately 2 TB of data over a 5–6 day run. However, about 70% of this is TIFF image data that can potentially be stored on tape after an analysis run is complete.

Depending on the application, single experiments run from 18–50 cycles. Paired-end experiments can double the number of cycles while gene expression experiments may use only 18-cycle protocols. Estimating required storage for individual runs depends on your application. The following table summarizes data volumes per experiment.

**Table 25** Data Volumes Per Experiment

Cycles per Run	Run Time (hours)	Raw Data (TB)	Results Data (TB)
18	42.0	0.360	0.270
26	60.7	0.520	0.390
36	84.0	0.720	0.540
50	116.7	1.000	0.750
75	175.0	1.500	1.125
100	233.3	2.000	1.500

### Network Infrastructure

These large data volumes mean that you will need:

1. A high-throughput ethernet connection (1 Gigabit or more recommended) or other data transfer mechanism.
2. A suitably large holding area for the images and analysis output (1 TB per run). As there will almost certainly some overlap between copying, analysis, possible reanalysis, 2–3 TB is an absolute minimum.
3. You need to consider which parts of the data you want to backup and what infrastructure you want to provide for the backup. If you want to keep image data, then half a terabyte per run is required. The Pipeline provides the option to perform loss-less data compression.

## Storage Configurations

You can configure your analysis server as either local storage or external network storage.

- ▶ Local server storage can be internal to the server, or Direct Attached Storage (DAS), which is a separate chassis attached to the server.
  - **Internal**—Simple but not scalable. Results data must be moved off to network storage at some point to make room for subsequent runs.
  - **DAS**—External chassis that is scalable since more than one DAS can be connected to the server. The server is an application server running the Pipeline and a file server providing access to results and receiving incoming raw data files.
- ▶ External network storage is either Network Attached Storage (NAS) or Storage Area Network (SAN). NAS and SAN are functionally equivalent, but SAN is larger, with higher performance, more connections, and more management options.
  - **NAS**—External chassis connected via an Ethernet to the server, instrument PC, and other clients on the network. NAS devices are scalable and highly optimized.
  - **SAN**—The most scalable with the highest performance. They have a very high bandwidth and support many simultaneous clients, but are complex to manage and significantly more expensive.

## Server Configurations

You can use either a single multi-processor, multi-core computer running Linux, or a cluster of Linux servers with a head node. The Pipeline can take advantage of clustered and multi-processing servers.

- ▶ **Single multi-processor, multi-core server**—Simple but not scalable. It can only analyze data from one Genome Analyzer, or two depending on power and your turn-around requirements.
- ▶ **Linux Cluster**—Highly scalable and capable of running multiple jobs simultaneously. It requires one server as a management node and a minimum number of computational nodes to be as efficient as a standalone server. By adding computational nodes, the cluster can service more instruments.

## Analysis Computer

The Pipeline may run on any Unix variant, if all of the prerequisites described in this section are met. However, Illumina does not support any platform other than Linux.

Illumina recommends and fully supports the following hardware configuration.

- ▶ High performance DL580 G4 server from Hewlett Packard  
This system comes configured with Red Hat Linux and the full installation of the Genome Analyzer Pipeline Software.
- ▶ Single 4-way, dual-core server with Xeon 7140 class processors
- ▶ 32 GB fault-tolerant RAM  
This is enough RAM to perform analysis tasks and file server tasks simultaneously. It uses high speed fault-tolerant hard drives for the operating system and applications.

▶ HP MSA20 Direct Attached Storage (DAS) unit

This capacity is intended to hold information from three runs, as follows:

- Last Processed Run—The results data from the last analyzed run are copied off to another storage server, where the run can be reviewed by the investigators and their staff. The raw image data is deleted.
- Currently Processed Run—The raw image data from the last completed instrument run are loaded and the Pipeline is performing analysis on that run.
- Next Run for Processing—The Genome Analyzer is copying the raw data from the current run up to the server.

As data volumes increase, the storage capacity can be scaled up by adding additional MSA20 DAS units.

On this type of hardware, you can expect to perform the image analysis and base calling for a full run in approximately one day. Sequence alignment takes additional time depending on which alignment program you are running; somewhere between a few hours (using our fast short-read whole-genome alignment program ELAND) and days (using more traditional alignment programs).

Pipeline parallelization is built around the multi-processor facilities of the “make” utility and scales very well to beyond eight nodes. Substantial speed increases are expected for parallelization across several hundred CPUs. For a detailed description, see *Using Parallelization* on page 93.



# Installation Prerequisites

The following software is required to run the Genome Analyzer Pipeline Software:

- ▶ Perl 5.8 or later; install the XML::Simple module and its dependencies (<http://www.cpan.org>)
- ▶ Python 2.3 or later
- ▶ GNU make 3.78 or later (qmake from Sun Grid Engine (SGE) 6.1 has been reported to work)
- ▶ gnuplot 3.7 or later (4.0 is recommended)
- ▶ ImageMagick 5.4.7 or later
- ▶ Ghostscript
- ▶ SMTP server (for optional automated email run reports)
- ▶ zlib
- ▶ bzlib

For a compilation from source, the following additional software is required:

- ▶ gcc (including g++)
- ▶ Optimized FFT library (Only one of the following three FFT libraries are required, not all three)
  - FFTW 3.0.1 or greater (3.1 is recommended); GPLed. To download files, see <http://www.fftw.org>.  
The single-precision version of FFTW is required (libfftw3f.a). This is produced by specifying the `--enable-single` option to the `./configure` procedure of FFTW as follows:  

```
./configure --enable-single
make
make install
```
  - Intel Maths Kernel Library
  - IBM ESSL

If you are running the Linux distribution Red Hat, the required dependencies listed above are satisfied by the Red Hat packages `perl-*`, `python-*`, `make`, `autoconf`, `gnuplot`, `ImageMagick`, `ghostscript`, `zlib`, `zlib-devel`, `bzip2`, `bzip2-devel`, `libtiff-devel` and `gcc-*` as well as their respective prerequisites. The Perl XML::Simple module and `fftw3` need to be downloaded separately and installed from source.

## Setting Up Email Reporting

The script `Gerald/runReport.pl` is called at the end of a run and sends you an email when a run successfully completes.

To use email notification, set up an SMTP server and set the following parameters in the GERALD configuration file. For additional information, see *GERALD Configuration File* on page 35.

1. Enter a space-separated list of the email addresses that should receive the run completion notification.

```
EMAIL_LIST your.name@domain.com that.name@domain.com
```

2. Indicate the path to the GERALD folder. The software assumes it can create a valid URL from the GERALD folder path by omitting the first two path elements and prepending WEB\_DIR\_ROOT.

```
WEB_DIR_ROOT http://server/SHARE
```

For example, if the path is /mnt/yourDrive/folder/folder/GERALD and WEB\_DIR\_ROOT is http://server/SHARE, the software will write the links as http://server/SHARE/folder/folder/GERALD/File.htm.

3. Identify your domain. Your SMTP server may refuse to accept emails from or send emails to addresses that do not end in @yourdomain.com.

```
EMAIL_DOMAIN yourdomain.com
```

4. Identify your IP address.

```
EMAIL_SERVER yourserver:2525
```

where yourserver is the name or IP address of a mail server that will accept SMTP email requests from you and 2525 is the port number of the SMTP service on that server.

Generally this will be 25. This is the default value if no port number is specified. The utility nmap, if installed, may help you identify which port on a server is hosting an SMTP service.

5. Test your email reporting by entering the following from the machine where you are running GERALD:

```
telnet yourserver yourPortNumber
```

If you don't get a friendly message, then email reporting will not work.

You can run runReport.pl directly in test mode by entering:

```
/runReport.pl --test yourserver:25 yourdomain.com  
anything your.name@yourdomain.com
```

You should receive a test email. If you do not, the transcript it generates should identify the problem.



The optional email reporting feature depends on how your SMTP servers are set up locally. Email reporting is not required to run the Pipeline to a successful completion.

# Installing the Pipeline Software

To install the Pipeline, you obtain the source code and then compile the software. Compiling the software will first build all C++ code, and then copy the relevant executables into the directories GOAT and GERALD, which contain the scripts and makefile generators.



If you want to use the Intel Math Kernel Library as an FFT backend, compile the image analysis module Firecrest separately from the rest of the project. Specify the additional variable MKL to make, as in "make MKL=true" and set the MKL-specific paths in the makefile to the appropriate locations on your system.

1. Go to the location where you want to install the Pipeline and type the following:

```
tar xvfz GAPipeline-version.tar.gz
```

where version is of the archive you have. You may have to adjust the path to the archive.

2. Change to the Pipeline directory and type:

```
make  
make install
```

## Compiling on Other Platforms

Compiling the Pipeline with the current makefiles works on all platforms, including many 32-bit and 64-bit Linux versions and Solaris. However, if your compilation does not succeed on a less commonly used platform (possibly 64-bit architectures or platforms other than Linux), you may have to make manual changes to the makefiles. Compilation problems, may require you to adapt the platform-specific gcc-compiler flags. Because of the optimized FFT libraries, the Firecrest makefile is particularly likely to be sensitive to platform-specific peculiarities.

Illumina does not support any platform other than Linux.

## Directory Setup

Create a directory called Instruments/<instrument\_name> for each Genome Analyzer in the same directory as the Run Folder, where <instrument\_name> is the hostname of the computer that is attached to the Genome Analyzer.

For example, the directory for the Run Folder /data/070813\_ILMN-1\_0217\_FC1234 would be called /data/Instruments/ILMN-1/.

If this directory exists, the Pipeline will place a file called default\_offsets.txt into this directory. The Pipeline automatically keeps this file up-to-date. For information on default\_offsets.txt, see *Image Offsets* on page 13.

Use the environment variable INSTRUMENT\_DIR, to override the default location of the Instruments directory:

```
export INSTRUMENT_DIR=/home/user/Instruments
```

If no instrument directory exists, the Pipeline will create one for you. If no default\_offsets.txt file exists, the Pipeline will create one with offset values equal to zero.



## Appendix B

# Output File Descriptions

### Topics

80	Introduction
80	Output File Types
81	Intensity Files
81	Sequence Files
82	Quality Score Files
82	Efficiency
83	Intermediate Output Data Files
86	Output File Formats
89	Parameters File Format

## Introduction

This section describes the file types and file formats of the intermediate data output produced during an analysis run.

## Output File Types

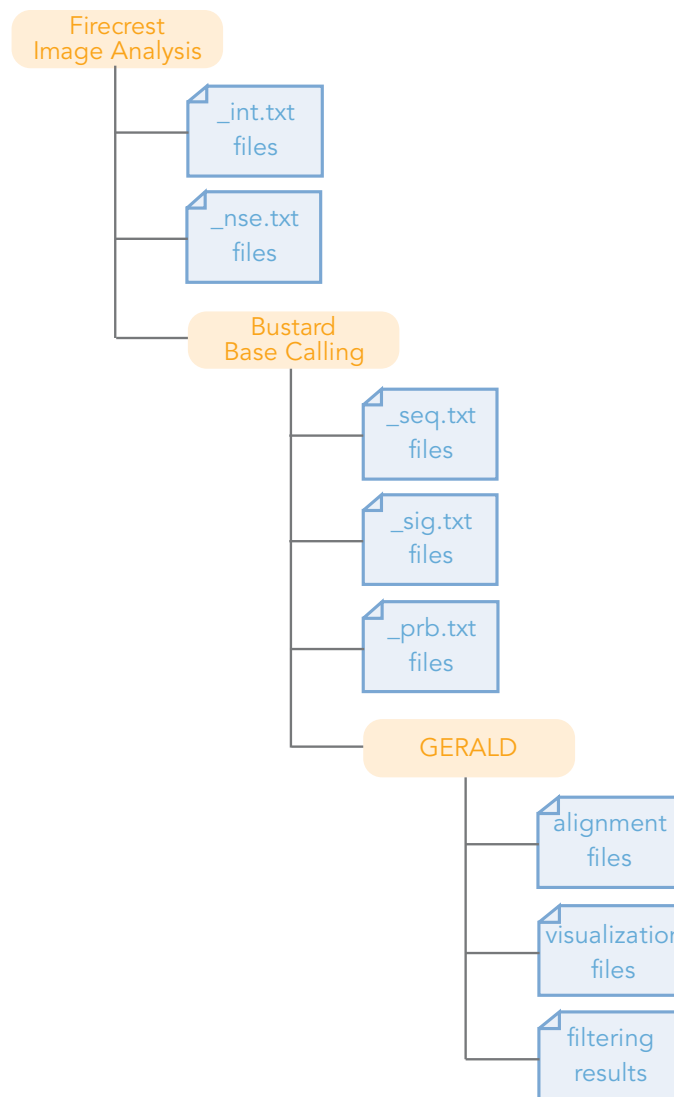


Figure 5 Run Folder Structure and Output File Types

## Intensity Files

The prefix of the intensity filenames follows the prefix of the image filenames, but the tile position is padded to four digits. For example, s\_1\_0006\_int.txt is the intensity file corresponding to the image files s\_1\_6\_a.tif, s\_1\_6\_c.tif, and so on.

Each intensity file has a set of data for remapped clusters on each line. Each row corresponds to the data from one cluster and each column is delimited by a space. Each row has a lane index, and a tile index in the first column, with the X offset and Y offset of the cluster in the second and third columns (all coordinates indexed from zero). These fields are tab-delimited. These values should be enough to uniquely identify any cluster for any run.

Following the coordinate fields are the data fields. The first value in a data field is the raw intensity for base A, the second is the raw intensity for base C, then G, and then T. These four values are separated by spaces and are followed by a tab to mark the beginning of the next four values. The next four values represent the corresponding intensities in the de-block scan (if a de-block scan has been performed), and then four intensities for the next cycle.

The number of fields should equal four coordinates plus four bases times the number of cycles of processed data, even if clusters don't yield data at the end of a set of cycles or part way through. In this case, the fields contain 0.0 and preserve delimiters.

The following is a sample line from an intensity file (\_int.txt):

```
<Channel><TAB><Tile><TAB><X><TAB><Y><TAB><A int 1st
      cycle> <C int> <G int> <T int><TAB>\
<A int 2nd cycle>... <LF>
```

A second set of files with an identical layout, stores the estimates of the noise on the intensity estimates. These files end in \_nse.txt.

## Sequence Files

The data found in the sequence files (\_seq.txt) located in the Bustard folder are raw sequences in the following condition:

- ▶ Trimming of any primer bases and splitting of a paired-read into two reads as specified by USE\_BASES has not been applied.
- ▶ Signal purity filtering of low quality data has not been applied.
- ▶ There is one file per tile, resulting in over 1000 files in total.

Use the sequence.txt files in the GERALD folder for which all the above points have been applied.

The base calls are kept in one file per tile for the concomitant base calls, and use the extension \_seq.txt. For a given intensity file, following base calling, we have a sequence file of the same name. For example, from an intensity file called s\_1\_0001\_int.txt you would get a base-called file named s\_1\_0001\_seq.txt.

Each sequence file has a sequence per row similar to the intensity files. Each row uses the same format as the intensity file, with the <lane>,<tile>,<X-offset>,<Y-offset> providing a unique key and a global co-ordinate for the sequence, and relating sequences to a cluster on the images. Following this format, the output is a string with one character for each base call in tab-delimited fields.

Another file holds the base caller confidence score that follows the format:

```
<channel><TAB><tile><TAB><X><TAB><Y><TAB><sequence><LF>
```

## Quality Score Files

Each intensity and sequence file has an associated file containing the quality scores for the base calls. For example, the quality file for the sequence file s\_1\_0002\_seq.txt is called s\_1\_0002\_prb.txt. There are four quality scores corresponding to the four bases in the order of A, C, G, T, for each base call in the sequence file. These fields are separated by a space. Each set of four scores for one base call is separated from the next set by a tab.

For following example might be the scores for a sequence AGT:

```
30 -30 -30 -30<TAB>-22 20 -27 -30<TAB>-17 17 -30 -30
```

The scores are defined as  $Q=-10*\log_{10}(p/(1-p))$ , where p is the probability of a base call corresponding to the base in question. By definition, the four values of p add up to 1.

Quality scores are essential for almost any genetic application to be able to tell good bases from bad bases. For example, ELAND can use quality scores to break degenerate alignments, and quality scores are essential for calling SNPs.

## Efficiency

To allow efficient handling by any software packages, there is one intensity and sequence file per tile. However, a single file can easily be created by simple concatenation of the individual files.



## Intermediate Output Data Files

Intermediate output files are found in the GERALD folder and contain data used to build the more meaningful results files described in *Analysis Output* on page 47.

The files are named using one of the following formats:

- ▶ s\_N\_TTTT\_name.txt, where N is the lane number, T is the tile number
- ▶ s\_N\_name.txt, where N is the lane number
- ▶ s\_N\_R\_name.txt, where N is the lane number, R is the read number

**Table 26** Intermediate Output File Descriptions

Output File	GERALD Analysis Mode	Description
s_N_TTTT_align.txt	ANALYSIS default ANALYSIS eland ANALYSIS monotemplate ANALYSIS eland_tag	Contains unfiltered first-pass alignments for a given tile.
s_N_TTTT_score.txt	ANALYSIS default ANALYSIS eland (if contaminant filtering is switched on)	Contains error rate information from first pass alignments. Error rate information is contained in text form and indicates potential contaminants. If CONTAM_FILE is specified, sequences with a negative entry in the s_N_cdifff.txt file, such as likely contaminants, are ignored.
s_N_TTTT_prealign.txt	ANALYSIS default ANALYSIS eland	Contains the realignment of all sequences against the data, using the error rate information in s_N_TTTT_score.txt to refine the alignment by re-weighting each base at each cycle according to its confidence.  If the lane in question were analyzed using ELAND, this file is just a copy of s_N_align.txt, because ELAND does not have the feature to weight the contribution of bases in an alignment.
s_N_TTTT_realign.txt	ANALYSIS default ANALYSIS eland	Consists of alignments in s_N_TTTT_prealign.txt, filtered to exclude alignments for those clusters that do not pass the quality criterion QF_PARAMS when applied to s_N_TTTT_qhg.txt.  Even if contaminant filtering is switched on, the alignments here will <b>not</b> have been contaminant filtered. Use the s_N_TTTT_credifff.txt file and qualityFilter.pl script to retain non-contaminants only. <pre>cat s_N_TTTT_realign.txt   qualityFilter.pl '(\$F[0]&gt;0)' s_N_TTTT_credifff.txt</pre> Replace ">" with "<=" to retain contaminants only.
s_N_TTTT_rescore.txt	ANALYSIS default ANALYSIS eland	Contains the improved estimate of the error rate based on s_N_TTTT_realign.txt. If CONTAM_FILE is specified, the calculation ignores sequences with a negative entry in the s_N_TTTT_credifff.txt file, such as likely contaminants.
s_N_TTTT_rescore.png	ANALYSIS default ANALYSIS eland	This generated image is a viewable error rate graph drawn from the data in s_N_TTTT_rescore.txt. This image is used as a thumbnail in Error.htm as described in <i>Visual Analysis Summary</i> on page 48.

Table 26 Intermediate Output File Descriptions (Continued)

Output File	GERALD Analysis Mode	Description
s_N_TTTT_qalign.txt	ANALYSIS default ANALYSIS monotemplate	Contains the alignments using base quality values to weight the bases. This file is not produced if the alignments for the lane in question were generated from an ELAND analysis, as ELAND does not have the feature to weight bases by their quality values.
s_N_TTTT_qraw.txt	ANALYSIS default ANALYSIS eland ANALYSIS monotemplate	This file collates the scores found in the file s_N_prb.txt, which contains four scores for each base. The highest of these scores is the score pertaining to the called base. If ANALYSIS --symbolic is specified (default), the quality scores are encoded as ASCII characters.
s_N_qraw.txt	ANALYSIS eland_extended	If ANALYSIS --numeric is specified, these are encoded as space separated integers.
s_N_R_qraw.txt	ANALYSIS eland_pair	In both cases the file will only contain values for cycles that the Pipeline has been asked to include, such as those with a "Y" in the corresponding USE_BASES string. For detailed descriptions of USE_BASES, see <i>USE_BASES Option</i> on page 31.
s_N_TTTT_qcal.txt	ANALYSIS default ANALYSIS eland ANALYSIS monotemplate	Contains quality values for each base, recalibrated using a calibration table derived from the alignments.
s_N_qcal.txt	ANALYSIS eland_extended	
s_N_R_qcal.txt	ANALYSIS eland_pair	
s_N_eland_query.txt	ANALYSIS eland_extended	Contains all reads for lane N and are concatenated into a single fasta file to use as an ELAND query.
s_N_R_eland_query.txt	ANALYSIS eland_pair	
s_N_eland_result.txt	ANALYSIS eland_extended	Contains the initial output of the ELAND alignment program run in the "standard" single-match mode.
s_N_R_eland_result.txt	ANALYSIS eland_pair	
s_N_eland_multi.txt	ANALYSIS eland_extended	Contains the initial output of the ELAND alignment program run in multiple-match mode.
s_N_R_eland_multi.txt	ANALYSIS eland_pair	
s_N_frag.txt	ANALYSIS eland_extended	Contains the alignment positions (based on at most 32 bases) and does an alignment of the full read to each position. A numeral refers to a run of matching bases, while an upper case base or N refers to a base in the reference that differs from the read.
s_N_eland_extended.txt	ANALYSIS eland_extended	Contains the corrected alignment positions and the full alignment descriptions for >32 base reads. This file is not purity filtered.
s_N_R_eland_extended.txt	ANALYSIS eland_pair	
s_N_saf.txt	ANALYSIS eland_extended	Short Alignment Format (SAF) aims to describe the best alignment for each read. The raw quality values are used to pick the best alignment from the (potentially) multiple possibilities.
s_N_R_saf.txt	ANALYSIS eland_pair	The software aims to pick the pair of alignments that is most consistent with the statistical distribution of insert sizes.

**Table 26** Intermediate Output File Descriptions (Continued)

Output File	GERALD Analysis Mode	Description
s_N_calsaf.txt	ANALYSIS eland_extended	These files are identical in format to s_N_saf.txt and s_N_R_saf.txt except the calibrated quality values are used to pick the best alignment.
s_N_R_calsaf.txt	ANALYSIS eland_pair	
s_N_qval.txt, s_N_qtable.txt	ANALYSIS default ANALYSIS eland	These are intermediate files produced during the generation of s_N_qcal.txt. Normally, they are deleted when the analysis is completed, but may be present in an analysis folder if the analysis was interrupted for any reason.

**Table 27** Contaminant Filtering-Specific Files

Output File	GERALD Analysis Mode	Description
s_N_TTTT_calign.txt	ANALYSIS default ANALYSIS eland (if contaminant filtering is switched on)	This file contains the first pass alignments of the sequences in the tile against the file of contaminant sequences specified in CONTAM_FILE. If CONTAM_FILE is not specified, this file is not produced.
s_N_TTTT_cdifff.txt	ANALYSIS default ANALYSIS eland (if contaminant filtering is switched on)	This file is only produced if CONTAM_FILE is specified. It contains the difference in alignment scores of alignment to data versus alignment to contaminant file. If negative, the corresponding sequence aligns better to contaminant than to data.
s_N_TTTT_crealign.txt	ANALYSIS default ANALYSIS eland (if contaminant filtering is switched on)	This file is only produced if CONTAM_FILE is specified. It contains realignments of the sequences in the tile against the file of contaminant sequences specified in CONTAM_FILE. The error rate information contained in s_N_score.txt refines the alignment by re-weighting each base at each cycle according to its confidence.
s_N_TTTT_cpredifff.txt	ANALYSIS default ANALYSIS eland (if contaminant filtering is switched on)	This file is only produced if CONTAM_FILE is specified. It contains differences in alignment scores of realignment to data from s_N_prealign.txt versus realignment to contaminant file s_N_crealign.txt. If the entry is negative, the corresponding sequence aligns better to contaminant than to data. This data is analogous to the data in s_N_cdifff.txt.
s_N_TTTT_credifff.txt	ANALYSIS default ANALYSIS eland (if contaminant filtering is switched on)	This file is only produced if CONTAM_FILE is specified. It contains differences in alignment scores of realignment to data from s_N_prealign.txt versus alignment to contaminant file s_N_crealign.txt, and is filtered to exclude alignments for those clusters that do not pass the quality criterion QF_PARAMS when applied to s_N_qhg.txt. This is based on s_N_cpredifff.txt, filtered to have a line-to-line correspondence with the realignments in s_N_realign.txt.

## Output File Formats

The sequences and base-specific quality scores are bundled by lane and come in several configurable text formats. The currently supported formats are fasta, fastq, and SCARF. For a description of each format, see *ANALYSIS Variables* on page 29.

Quality scores are stored as either symbolic ASCII values or numeric values. The parameters that set the configuration of the output format are described in *ANALYSIS Variables* on page 29.

**Table 28** Final Output File Formats

Output File	Format
s_N_export.txt s_N_R_export.txt	<p>Not all fields are relevant to a single-read analysis.</p> <ol style="list-style-type: none"> <li>1. Machine (Parsed from Run Folder name)</li> <li>2. Run Number (Parsed from Run Folder name)</li> <li>3. Lane</li> <li>4. Tile</li> <li>5. X Coordinate of cluster</li> <li>6. Y Coordinate of cluster</li> <li>7. Index string (Blank for a non-indexed run)</li> <li>8. Read number (1 or 2 for paired-read analysis, blank for a single-read analysis)</li> <li>9. Read</li> <li>10. Quality string—In symbolic ASCII format (ASCII character code = quality value + 64) by default (Set QUALITY_FORMAT --numeric in the GERALD config file for numeric values)</li> <li>11. Match chromosome—Name of chromosome match OR code indicating why no match resulted</li> <li>12. Match Contig—Gives the contig name if there is a match and the match chromosome is split into contigs (Blank if no match found)</li> <li>13. Match Position—Always with respect to forward strand, numbering starts at 1 (Blank if no match found)</li> <li>14. Match Strand—"F" for forward, "R" for reverse (Blank if no match found)</li> <li>15. Match Descriptor—Concise description of alignment (Blank if no match found) <ul style="list-style-type: none"> <li>• A numeral denotes a run of matching bases</li> <li>• A letter denotes substitution of a nucleotide: For a 35 base read, "35" denotes an exact match and "32C2" denotes substitution of a "C" at the 33rd position</li> </ul> </li> <li>16. Single-Read Alignment Score—Alignment score of a single-read match, or for a paired read, alignment score of a read if it were treated as a single read (Blank if no match found)</li> <li>17. Paired-Read Alignment Score—Alignment score of a paired read and its partner, taken as a pair (Blank for single-read analysis)</li> <li>18. Partner Chromosome—Name of the chromosome if the read is paired and its partner aligns to another chromosome (Blank for single-read analysis)</li> <li>19. Partner Contig—Not blank if read is paired and its partner aligns to another chromosome and that partner is split into contigs (Blank for single-read analysis)</li> <li>20. Partner Offset—If a partner of a paired read aligns to the same chromosome and contig, this number, added to the Match Position, gives the alignment position of the partner (Blank for single-read analysis)</li> <li>21. Partner Strand—To which strand did the partner of the paired read align? "F" for forward, "R" for reverse (Blank if no match found, blank for single-read analysis)</li> <li>22. Filtering—Did the read pass quality filtering? "Y" for yes, "N" for no</li> </ol>

Table 28 Final Output File Formats (Continued)

Output File	Format
s_N_sequence.txt s_N_R_sequence.txt	Filtered output User-specified: fasta, fastq, scarf (one sequence per line, not identifier)
s_N_TTTT_realign.txt	Final quality-filtered sequence alignments Space-separated text values: <ol style="list-style-type: none"> <li>1. sequence</li> <li>2. best score</li> <li>3. number of hits at that score</li> </ol> The following columns only appear if hits equal 1 (a single, unique match) <ol style="list-style-type: none"> <li>4. target:pos</li> <li>5. strand</li> <li>6. target sequence</li> <li>7. next best score</li> </ol>
s_N_rescore.txt	Estimate of the error rate based on s_N_TTTT_realign.txt Tabular text format, header data included

Table 29 Intermediate Output File Formats

Output File	Format
s_N_eland_results.txt s_N_R_eland_results.txt	Unfiltered ELAND alignment output Each line of the output file contains the following fields: <ol style="list-style-type: none"> <li>1. Sequence name (derived from file name and line number if format is not fasta)</li> <li>2. Sequence</li> <li>3. Type of match codes: <ul style="list-style-type: none"> <li>• NM—No match found</li> <li>• QC—No matching done: QC failure (too many Ns)</li> <li>• RM—No matching done: repeat masked (may be seen if repeatFile.txt was specified)</li> <li>• U0—Best match found was a unique exact match</li> <li>• U1—Best match found was a unique 1-error match</li> <li>• U2—Best match found was a unique 2-error match</li> <li>• R0—Multiple exact matches found</li> <li>• R1—Multiple 1-error matches found, no exact matches</li> <li>• R2—Multiple 2-error matches found, no exact or 1-error matches</li> </ul> </li> <li>4. Number of exact matches found</li> <li>5. Number of 1-error matches found</li> <li>6. Number of 2-error matches found</li> <li>7. The following fields are only used if a unique best match was found:</li> <li>8. Genome file in which match was found</li> <li>9. Position of match (bases in file are numbered starting at 1)</li> <li>10. Direction of match (F=forward strand, R=reverse)</li> <li>11. How N characters in read were interpreted (". "=not applicable, "D"=Detection, "I"=Insertion)</li> </ol> The following field is only used in the case of a unique inexact match: <ol style="list-style-type: none"> <li>12. Position and type of first substitution error (A numeral refers to a run of matching bases, an upper case base or N refers to a base in the reference that differs from the read. For example, 11A: after 11 matching bases, base 12 is A in the reference but not in the read)</li> </ol>

Table 29 Intermediate Output File Formats (Continued)

Output File	Format
s_N_eland_multi.txt s_N_R_eland_multi.txt	<p>Each line of the output file contains the following fields:</p> <ol style="list-style-type: none"> <li>1. Sequence name</li> <li>2. Sequence</li> <li>3. Either NM, QC, RM (as described above) or the following:</li> <li>4. x:y:z where x, y, and z are the number of exact, single-error, and 2-error matches found</li> <li>5. Blank, if no matches found or if too many matches found, or the following: BAC_plus_vector.fa:163022R1,170128F2,E_coli.fa:3909847R1 This says there are two matches to BAC_plus_vector.fa: one in the reverse direction starting at position 160322 with one error, one in the forward direction starting at position 170128 with two errors. There is also a single-error match to E_coli.fa.</li> </ol>
s_N_TTTT_align.txt	<p>Unfiltered first-pass alignments</p> <p>Each line of the output file contains the following fields:</p> <ol style="list-style-type: none"> <li>1. Sequence</li> <li>2. Best score</li> <li>3. Number of hits at that score</li> </ol> <p>The following columns only appear if hits equal 1 (a single, unique match)</p> <ol style="list-style-type: none"> <li>4. Target:pos</li> <li>5. Strand</li> <li>6. Target sequence</li> <li>7. Next best score</li> </ol>
s_N_TTTT_prealign.txt	<p>Unfiltered second-pass alignments</p> <p>Each line of the output file contains the following fields:</p> <ol style="list-style-type: none"> <li>1. Sequence</li> <li>2. Best score</li> <li>3. Number of hits at that score</li> </ol> <p>The following columns only appear if hits equal 1 (a single, unique match)</p> <ol style="list-style-type: none"> <li>4. Target:pos</li> <li>5. Strand</li> <li>6. Target sequence</li> <li>7. Next best score</li> </ol>

## Parameters File Format

The top level Run Folder contains a parameters file, named <Run Folder Name>.params, and is written in the following format:

```
<experiment>
  <run>
  ...
</run>
<run>
  ...
</run>
</experiment>
```

For each restart of the instrument, a new run tag with corresponding parameter tags is added to the parameters file. For most experiments, there will only be one run.

The XML tags in the parameters file are self-explanatory. The following shows an example of a parameters file:

```
<experiment>
  <run>
    <instrument>slxa-b1</instrument>
  </run>
</experiment>
```

In the top level of the Data folder you will find the parameters file that records any information specific to the generation of the subfolders. This contains a tag-value list describing the cycle-image folders used to generate each folder of intensity and sequence files.

```
<?xml version="1.0"?>
<ImageAnalysis>
  <Run Name="C1-24_Firecrest1.9.0_30-07-2007_user">
    <Cycles First="1" Last="24" Number="24" />
    <ImageParameters>
      <AutoOffsetFlag>1</AutoOffsetFlag>
      <AutoSizeFlag>0</AutoSizeFlag>
      <DataOffsetFile>/data/070813_ILMN-1_0217_FC1234/
        Data/default_offsets.txt</DataOffsetFile>
      <Fwhm>2.700000</Fwhm>
      <InstrumentOffsetFile></InstrumentOffsetFile>
      <OffsetFile>/data/070813_ILMN-1_0217_FC1234/Data/
        default_offsets.txt</OffsetFile>
      <Offsets X="0.000000" Y="0.000000" />
      <Offsets X="0.790000" Y="-0.550000" />
      <Offsets X="-0.240000" Y="-0.140000" />
      <Offsets X="0.190000" Y="0.650000" />
      <RemappingDistance>1.500000</RemappingDistance>
      <SizeFile></SizeFile>
      <Threshold>4.000000</Threshold>
    </ImageParameters>
    <RunParameters>
      <AutoCycleFlag>0</AutoCycleFlag>
      <BasecallFlag>1</BasecallFlag>
      <Compression>gzip</Compression>
      <CompressionSuffix>.gz</CompressionSuffix>
      <Deblocked>0</Deblocked>
```

```

<DebugFlag>0</DebugFlag>
<ImagingReads Index="1">
  <FirstCycle>1</FirstCycle>
  <LastCycle>24</LastCycle>
  <RunFolder>/data/070813_ILMN-1_0217_FC1234</
    RunFolder>
</ImagingReads>
<Instrument>ILMN-1</Instrument>
<MakeFlag>1</MakeFlag>
<MaxCycle>-1</MaxCycle>
<MinCycle>-1</MinCycle>
<Reads Index="1">
  <FirstCycle>1</FirstCycle>
  <LastCycle>24</LastCycle>
  <RunFolder>/data/070813_ILMN-1_0217_FC1234</
    RunFolder>
</Reads>
<RunFolder>/data/070813_ILMN-1_0217_FC1234</
  RunFolder>
<Software Name="Firecrest" Version="1.9.0" />
<TileSelection>
  <Lane Index="8">
    <Sample>s</Sample>
    <Tile>10</Tile>
    <Tile>20</Tile>
    <Tile>30</Tile>
  </Lane>
</TileSelection>
<Time>
  <Start>30-07-07 12:50:45 BST</Start>
</Time>
<User Name="user" />
</Run>
<Run Name="Cl-24_Firecrest1.9.0_30-07-2007_user.2">
  ...
</Run>
</ImageAnalysis>

```

In each image analysis folder there is another parameters file containing the meta-information about the base caller runs.

```

<?xml version="1.0"?>
<BaseCallAnalysis>
  <Run Name="Bustard1.9.0_30-07-2007_user">
    <BaseCallParameters>
      <Matrix Path="">
        <AutoFlag>1</AutoFlag>
        <AutoLane>0</AutoLane>
        <Cycle>2</Cycle>
        <FirstCycle>1</FirstCycle>
        <LastCycle>24</LastCycle>
        <Read>1</Read>
      </Matrix>
      <MatrixElements />
      <Phasing Path="">
        <AutoFlag>1</AutoFlag>
        <AutoLane>0</AutoLane>
        <Cycle>1</Cycle>

```



```
<FirstCycle>1</FirstCycle>
<LastCycle>24</LastCycle>
<Read>1</Read>
</Phasing>
<PhasingRestarts />
</BaseCallParameters>
<Cycles First="1" Last="24" Number="24" />
<Input Path="C1-24_Firecrest1.9.0_30-07-
2007_user.2" />
<RunParameters>
  <AutoCycleFlag>0</AutoCycleFlag>
  <BasecallFlag>1</BasecallFlag>
  <Compression>gzip</Compression>
  <CompressionSuffix>.gz</CompressionSuffix>
  <Deblocked>0</Deblocked>
  <DebugFlag>0</DebugFlag>
  <ImagingReads Index="1">
    <FirstCycle>1</FirstCycle>
    <LastCycle>24</LastCycle>
    <RunFolder>/data/070813_ILMN-1_0217_FC1234</
    RunFolder>
  </ImagingReads>
  <Instrument>ILMN-1</Instrument>
  <MakeFlag>1</MakeFlag>
  <MaxCycle>-1</MaxCycle>
  <MinCycle>-1</MinCycle>
  <Reads Index="1">
    <FirstCycle>1</FirstCycle>
    <LastCycle>24</LastCycle>
    <RunFolder>/data/070813_ILMN-1_0217_FC1234</
    RunFolder>
  </Reads>
  <RunFolder>/data/070813_ILMN-1_0217_FC1234</
  RunFolder>
</RunParameters>
<Software Name="Bustard" Version="1.9.0" />
<TileSelection>
  <Lane Index="5">
    <Sample>s</Sample>
    <TileRange Max="5" Min="5" />
  </Lane>
</TileSelection>
<Time>
  <Start>30-07-07 18:01:50 BST</Start>
</Time>
<User Name="user" />
</Run>
</BaseCallAnalysis>
```





## Appendix C

# Using Parallelization

### Topics

- 94 Introduction
- 94 “Make” Utilities
  - 94 Standard “Make”
  - 94 Customizing Parallelization
  - 94 Distributed “Make”
- 97 Parallelization Limitations
- 98 Memory Limitations

## Introduction

One of the main considerations behind the current Pipeline architecture is the ability to use the parallelization facilities present on almost all SMP machines and on most Linux/Unix clusters. Parallelization is scalable and makes use of all available CPU power.

## “Make” Utilities

Parallelization is built around the ability of the standard “make” utility to execute in parallel across multiple processes on the same computer. Since version 0.2.2, the Pipeline also provides a series of checkpoints and hooks that enables you to customize the parallelization for your computing setup. See *Customizing Parallelization* on page 94 for details.

### Standard “Make”

The standard “make” utility has many limitations, but it is universally available and has a built-in parallelization switch (“-j”). For example, on a dual-processor, dual-core system, running “make -j 4” instead of “make,” executes the Pipeline run in parallel over four different processor cores, with an almost 4-fold decrease in analysis run time. On a 4-way SMP system, “-j 8” or more may be advisable.

### Distributed “Make”

There are several distributed versions of “make” for cluster systems. Frequently used versions include “qmake” from Sun Grid Engine and “lsmake” from LSF.

To use “qmake,” a short wrapper script is required. See the grid engine documentation for details.

There are known issues with the use of “lsmake” that prevent parts of the Pipeline from running. Therefore, Illumina does not recommend using “lsmake” to run the Pipeline.



Distributed cluster computing may require significant system administration expertise. Illumina does not support external installations.

### Customizing Parallelization

Many parts of the Pipeline are intrinsically parallelizable by lane or tile. However, some parts of the Pipeline cannot be parallelized completely. Pipeline v.0.2.2 and later, has a series of additional hooks and check-points for customization.

The Pipeline workflow is divided into the image analysis, base calling, alignment. You can divide it further into a series of steps with different levels of scalability where synchronization “barriers” cause the Pipeline to wait for each of the tasks within a step to finish before going to the next step.

You can parallelize the steps at the run level (no parallelization), the lane level (up to eight jobs in parallel), and the tile level (up to thousands of jobs in parallel). Each step is initiated by a "make" target. After completion of each of these steps, the Pipeline produces a file or a series of files at the lane/tile level, that determines whether all jobs belonging to the step have finished. Finally, hooks are provided upon completion of the step to issue user-defined external commands.

The Firecrest makefile creates two files, lanes.txt and tiles.txt, containing a list of all lanes and tiles used in the run. This information is parsed and used to feed your own analysis scripts.

## Example of Parallelization

Typing "make" in the Firecrest folder is equivalent to the following series of commands:

```
make default_offsets.txt
make s_1; make s_2; make s_3; make s_4; make s_5;
      make s_6; make s_7; make s_8
make all
```

This command addresses each lane sequentially. Using parallelization, you can run all eight commands on the second line in parallel, as long as you make sure that they all finish before the final "make all" is issued. There are several ways to parallelize these jobs. For example, you could send them to the queue of a batch system, or just use "ssh" or "rsh" to send them to a predetermined analysis computer.

In the following example, the second step is automatically started after the first step (make s\_1;) as the external command, "cmdf1." The external command will be issued after completion of the first step.

```
make -j 2 default_offsets.txt cmdf1='make s_1;
      make s_2; make s_3; make s_4; \
make s_5; make s_6; make s_7; make s_8;' \
cmdf2='if [[ -e s_1_finished.txt && -e
      s_2_finished.txt && -e s_3_finished.txt \
&& -e s_4_finished.txt && -e s_5_finished.txt
&& -e s_6_finished.txt \
&& -e s_7_finished.txt && -e s_8_finished.txt ]]; then
      make all ; fi #'
```

This only makes sense if you parallelize the eight "make" commands instead of using "make s\_1," as shown in the following example:

```
nohup ssh <mycomputenodel> make -j 4 s_1
```

—or—

```
bsub make s_1
```

After completing the eight "make" commands in the second step, the shell command "cmdf2" is run to check for the existence of all eight checkfiles. The next make command (make all) will be issued only after the completion of the first seven lanes.

```
if [[ -e s_1_finished.txt && -e s_2_finished.txt
&& -e s_3_finished.txt \
&& -e s_4_finished.txt && -e s_5_finished.txt
&& -e s_6_finished.txt \
&& -e s_7_finished.txt && -e s_8_finished.txt ]];
then make all ; fi #
```

The reason for the final comment symbol (#) at the end of the shell command above is that the Pipeline automatically supplies an argument to all commands issued at the lane level and is used as an identifier for the actual lane analyzed. In the example above, this argument is not used, and so it needs to be commented out.



There is no need to declare the full shell command on the command line. You could put all of the shell commands into a shell script and call that script instead.

## Image Analysis

This section lists the steps, corresponding make targets, checkfiles, and hooks for image analysis by the Firecrest module.

Parallelization Level	Run	Lane	Tile
Target	default_offsets.txt		
Check File	default_offsets.txt		
Hook	cmdf1		
Target		s_1	s_1_0001
Check File		s_1_finished.txt	(none)
Hook		cmdf2	(none)
Target	all		
Check File	finished.txt		
Hook	cmdf3		

## Base Calling

This section lists the steps, corresponding make targets, checkfiles, and hooks for base calling by the Bustard module.

Parallelization Level	Run	Lane	Tile
Target		Phasing/ s_1_phasing.xml	Phasing/ s_1_0001_phasing.txt
Check File		Phasing/ s_1_phasing.xml	Phasing/ s_1_0001_phasing.txt
Hook		cmdb1	(none)
Target	Phasing/ phasing.xml		

Parallelization Level	Run	Lane	Tile
Check File	Phasing/ phasing.xml		
Hook	cmdb2		
Target		s_1	s_1_0001
Check File		s_1_finished.txt	s_1_0001_qhg.txt
Hook		cmdb3	(none)
Target	all		
Check File	finished.txt		
Hook	cmdb4		

### Sequence Alignment

This section lists the steps, corresponding make targets, checkfiles and hooks for sequence alignment by the GERALD module.

Parallelization Level	Run	Lane
Target		s_1
Check File		s_1_finished.txt
Hook		(none)
Target	all	
Check File	finished.txt	
Hook	POST_RUN_COMMAND (Accessible from GERALD config file)	

### Parallelization Limitations

The analysis works on a per-tile basis, so the maximum degree of parallelization achievable is equal to the total number of tiles scanned during the run. However, some parts of the Pipeline operate on a per-lane basis, and a few parts on a per-run basis, which means that scaling will cease to be linear at some stage for more than 8-way parallelization.

## Memory Limitations

Most parts of the Pipeline have moderate memory requirements. However, very dense runs on a Genome Analyzer II can require up to 2 GB per process. ELAND uses up to 1 GB, which means that parallelization of ELAND is more likely to run into memory issues. Because many load-sharing systems do not take into account the memory used, ELAND is treated differently in the Pipeline. Its parallelization is artificially prevented by a non-essential “make” dependency. If you are certain that you cannot exhaust your available memory, you can use a special option to the GERALD configuration file (ELAND\_MULTIPLE\_INSTANCES 8) to remove this dependency. However, you are responsible for making sure that you have up to 8 GB of RAM at your disposal. For additional information, see *Using GERALD* on page 27.





## Appendix D

# Base Call Calibration and Alignment Scoring

### Topics

100	Introduction
100	Goal
100	Method
100	Modifications to the Phred Formula
101	Characteristics of the Quality Scores Produced by the Base Caller
101	High Quality Scores
101	Limitations of the Recalibration Method
101	Major Alignment Errors
101	SNP Rate
101	Size of Data Set
102	General Usage
102	Configuring Quality Table Sources in GERALD
104	Expert Usage
104	Extracting Quality Predictors
106	Extracting Reference Bases
107	Creating a Quality Table
109	Generating New Quality Values
111	Configuring Quality Table Sources in GERALD
112	Default and Experimental Predictors
113	Further Considerations
114	Frequently Asked Questions

## Introduction

The Genome Analyzer Pipeline Software contains a module for quality score calibration. This is essentially a re-implementation of the method described in the phred paper by Ewing & Green (Ewing B, Green P (1998) Base-Calling of Automated Sequencer Traces Using Phred. II. Error Probabilities. *Genome Research*, 8, 186).

**Goal** The main goal of the quality calibration is to bring the quality scores and corresponding predicted error rates that the base-caller generates into line with the error rates obtained from an alignment.

**Method** The calibration framework uses a set of trace-specific and base-specific parameters that are indicative of the base-call quality to predict a new calibrated quality score. The mapping between old and new scores is encoded in a phred table. The mapping is calibrated on a set of known alignments. The Pipeline performs the calibration procedure as a post-processing step to the base-calling.

Each lane can be calibrated separately. The alignment obtained from the lane itself can be used as a training set, resulting in a procedure we refer to as auto-calibration (e.g. training set and target data are the same). In addition, a table derived from a different data set (or a control lane) can be applied (cross-calibration).

**Modifications to the Phred Formula** The original base-caller (Bustard) scores are based on an error propagation of the estimated noise on the underlying raw cluster intensities. The base-caller generates four quality scores rather than just one, given the relative probabilities that the underlying read is an A, C, G or T. The scores are captured in a file format called `_prb.txt`. See *Understanding the Run Folder* on page 8.

In order to accommodate the four different reads and obtain a sensible dynamic range, we have modified the phred formula slightly to generate a new set of quality scores that approaches the phred definition asymptotically for  $Q > 10-15$ .

# Characteristics of the Quality Scores Produced by the Base Caller

The raw scores produced by the base-caller have the following characteristics:

1. They are monotonic predictors of base-quality even out to nominal Q scores out to at least 40.
2. They provide additional information even for the non-called base.

## High Quality Scores

The calibration may be far from perfect, particularly for high quality scores  $Q \gg 10$ . Their dynamic range is artificially limited (currently to Q40). This is an arbitrary cut-off motivated by the fact that the quality estimation procedure used by the base-caller is unlikely to be accurate beyond the cut-off.

## Limitations of the Recalibration Method

The aim of the recalibration is to improve the correlation of the scores with the error rates obtained from alignment against a known reference. There are some limitations to this method as described in this section.

### Major Alignment Errors

The auto-calibration procedure that can be applied to improve the quality calibration is based on the assumption that the alignments are more or less correct; violations of that assumption will skew the calibration and limit the accuracy of the calibration. For example, if the sample in question contains a contaminant (e.g. *E. coli* sequence) at the 5% level, the corresponding reads (depending on read-length and target genome) may be mistaken as aligned reads and contribute significantly to the error rates.

### SNP Rate

A reference obtained for a different individual would limit the maximum quality to the rate at which SNPs are observed, since the quality scores cannot get better than the SNP rate.

### Size of Data Set

The maximum alignment score is also limited by the size of the data set. For example, in order to obtain Q40, you would need around  $10^4$  base pairs at a quality of Q40 (presumably, even more because of the Poisson counting error).

## General Usage

The calibration is part of the Pipeline version 0.2 and forward, and will be run automatically as part of the genomic analysis. In Pipeline version 1.0, the calibration has been rewritten to allow calibration from an external table or a control lane.

### Configuring Quality Table Sources in GERALD

By default, the Pipeline generates a quality table for each lane (or for each read in the lane) in which an analysis including alignment is performed and then uses this quality table (or pair of quality tables) to reestimate the base-call quality values of all the tiles in that lane.

#### Configuring Quality Table Sources

The source of the quality table used in the quality calibration for a lane may be overridden by defining `QCAL_SOURCE` (or, for individual reads within a paired read analysis, `QCAL_SOURCE1` and/or `QCAL_SOURCE2`) in the `config.txt` passed to `GERALD.pl` to configure GERALD analysis.

The supported values of the `QCAL_SOURCE` variables are listed below:

**Table 30** *QCAL\_SOURCE Variable Values*

Value	Description
auto	The qtable(s) used within the lane (to reestimate the base-call quality values) are the qtable(s) generated for that lane (from the quality predictor values and called and reference base values of bases in reads from that lane)
auto<n>, where n is the number of a lane for which alignment will be performed	The qtable(s) used in the lane are those generated for lane n. For example, "auto5" means that the qtable(s) from lane 5 are used
/path/to/qtable.txt	The qtable file at the specified path is used.



#### NOTE

As with any `config.txt` variable, the `QCAL_SOURCE` variables may be specified for all lanes of a flow cell or for any non-overlapping subsets of the flow cell lanes, with the latter overriding the former, if both are specified.

### Paired-End Analysis

For cases where paired-end analysis is in use, the following principles apply:

1. If both lanes are paired, then any specification of the source lane for read 1 of the target lane results in the read 1 qtable of the source lane being used as the read 1 qtable in the target lane, and similarly for read 2.
2. If only the target lane is paired, then there is only one qtable available in the source lane but it may be used for both reads in the target lane.
3. If only the source lane is paired, then its read 1 qtable is (arbitrarily) used.



In a paired-read analysis lane, specification of QCAL\_SOURCE1 and/or QCAL\_SOURCE2 will override specification of QCAL\_SOURCE, although the latter will be used if it has been specified and not overridden for a given read.

## Example

The following example specifies the following condition:

1. Lanes 1–3 read 1 will use the lane 8 read 1 qtable; lanes 1–3 read 2 will use the lane 8 read 2 qtable
2. Lane 4 read 1 will use the external ref51\_qtable.txt qtable
3. Lane 4 read 2 will use the lane 7 read 2 qtable
4. Lanes 5–8 reads 1 and 2 will use the ref42\_qtable.txt qtable

```
ANALYSIS eland_pair
QCAL_SOURCE /home/illumina/ref42_qtable.txt
123:QCAL_SOURCE auto8
4:QCAL_SOURCE1 /home/illumina/ref51_qtable.txt
4:QCAL_SOURCE2 auto7
```

Note that even though the lane 8 qtables will not be needed in lane 8, they will still be generated for use in lanes 1–3.

## Expert Usage

### Extracting Quality Predictors

This section describes the extraction of quality predictors.

The following is a simple example of extracting base-call quality predictor (qval files):

```
Pipeline/QCalibration/extract_quality_predictors  
s_2_0059_sig2.txt s_2_0059_prb.txt  
s_2_0059_seq.txt s_2_0059_qval.txt
```

### Input Data

Input data is read from the following files:

- ▶ sig2 files containing the per-channel intensities.
- ▶ prb files containing the uncalibrated probabilities of each possible base type for each base.
- ▶ seq files containing the called bases.

### Output Data

The predictor values are written to the qval file.

### Compression

Either or both of the relatively large sig2 and qval files may be compressed and contain an additional suffix in the file name. The QVAL\_COMPR definition in the GERALD makefile specifies which type of compression, if any, should be used.

### Predictors

The current default predictors are as follows:

- ▶ Cycle number
- ▶ Per-base purity score
- ▶ Minimum purity score over the first PURE\_BASES cycles
- ▶ Raw base-caller quality score

The choice of predictors is configurable.

### Format of the qval File

The format of the uncompressed qval file is as follows:

- ▶ Each row corresponds to one base.
- ▶ The columns are tab-separated and each contains the values for one of the quality predictors, with the exception of the last column, which contains the called base types.

### Paired-End Analysis

The above example is for a single-read analysis. For a paired-read analysis, the following must be modified:

- ▶ The “--multi\_read” option must be specified.
- ▶ The “--orig\_read\_lengths” option must be specified and also have as its value the colon-separated lengths of the two reads, as in the following example:
 

```
--orig_read_lengths 36:36
```
- ▶ Two qval files must be specified, one each for the predictor values associated with the two reads.

## Detailed Usage of extract\_quality\_predictors

The detailed usage of the extract\_quality\_predictors binary is as follows:

- ▶ For information about usage:
 

```
./extract_quality_predictors -h|--help
```
- ▶ For extracting quality predictors (with available options listed):
 

```
./extract_quality_predictors
[-l|--orig_read_lengths <ORIG_READ_LENGTHS>
 [-m|--multi_read]]
[-p|--pure_bases <PURE_BASES>]
[-z|--sig2_compression <gzip|bzip2>]
[-Z|--qval_compression <gzip|bzip2>]
[-x|--predictors <PREDICTOR_LIST>]
<sig2_file> <prb_file> <seq_file>
<qval_file | read1_qval_file read2_qval_file ...>
```

The format of the variables is described below:

Variable	Description
ORIG_READ_LENGTHS	Colon-separated list
PURE_BASES	The number of cycles over which max_early_unchastity is calculated (default 12)
PREDICTOR_LIST	Colon-separated list drawn from the following valid predictor names: <ul style="list-style-type: none"> <li>• homopol_len</li> <li>• in_read_cycle</li> <li>• max_early_unchastity</li> <li>• max_local_unchastity</li> <li>• raw_unquality</li> <li>• signal_decay</li> <li>• unchastity</li> </ul> <p><b>Note:</b> The default value for PREDICTOR_LIST is            in_read_cycle:unchastity:max_early_unchastity:raw_unquality</p>



### NOTE

The naming of the predictors reflects the fact that the prediction algorithm expects higher predictor values to correspond to lower base-call quality. For example, “unchastity” is simply “1 - chastity.”

## Extracting Reference Bases

In addition to the information contained in the qual files, the calibration process also requires the reference value for each base for which it is available; this is the case when the base is within a read that has been uniquely aligned.



In versions of the Pipeline prior to version 1.0, these reference base values were included in the qual files; the current separation reflects the stages within the Pipeline at which the two types of information become available.

### Method

The called bases are derived from the combination of seqpre files with saf files (or from the corresponding align files that are produced by older analysis modes) using saf2qref.pl Perl script (or align2qref.pl for older analysis modes).

### Detailed Usage of saf2qref.pl

The usage of the saf2qref.pl script is as follows:

```
paste seqpreFile safFile | ./saf2qref.pl
[--read1 | --read2]
--use_bases USE_BASES
--orig_read_lengths ORIG_READ_LENGTHS
qref_prefix qref_suffix > qrefFile
```



The qref\_prefix and qref\_suffix options are required so that saf2qref.pl can generate a qref file for each tile represented in the per-lane saf file. The qref\_prefix should normally be the lane prefix plus a trailing underscore. For example, s\_1\_ is the prefix for lane 1. The qref suffix will normally be \_qref.txt for single-read analysis, and \_<READ\_NUM>\_qref.txt, (\_1\_qref.txt) for read 1, for paired-read analysis.

The format of the variables is described below:

Variables	Description
USE_BASES	<p>The USE_BASES string should correspond in length to the total number (across all reads) of cycles passed through from the Bustard analysis module and contain only the following:</p> <ul style="list-style-type: none"> <li>• "Y" at the position of a cycle to be included in read 1 analysis</li> <li>• "y" at the position of a cycle to be included in read 2 analysis</li> <li>• "n" at the position of any cycle to be masked out of analysis</li> </ul> <p>Occurrences of "y" must follow occurrences of "Y." In addition, only contiguous occurrences of "Y" and "y" respectively are supported.</p>



Variables	Description
ORIG_READ_LENGTHS	The ORIG_READ_LENGTHS string should comprise a colon-separated list of the original lengths of reads passed through from the Bustard analysis stage. For paired reads, this might be "36:36." The sum of these values should equal the length of the USE_BASES string.

## Paired-End Analysis

For paired-read analysis, either "--read1" or "--read2" option must be specified as appropriate.

## Example

The following is a complete example:

```
paste s_1_1_seqpre.txt s_1_1_saf.txt |
    Pipeline/Gerald/saf2qref.pl --read1
--use_bases
    YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYnYYYYYYY
    YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYn
--orig_read_lengths 36:36 s_1_1_qref.txt
```

## Older Analysis Modes

The usage of align2qref.pl for older analysis modes is somewhat simpler as none of the older modes support paired reads and the align files are per tile (note that align2qref.pl writes all the reference bases it extracts to a standard output, unlike saf2qref.pl which generates an output file per tile).

The usage of the align2qref.pl script for older analysis modes is as follows:

```
cat alignFile | ./align2qref.pl
    --use_bases USE_BASES > qrefFile
```

## Output File Format

The format of the qref file produced with either analysis mode is the same, a single column of reference bases. Within sets of bases corresponding to an aligned read, any bases corresponding to cycles masked out by USE\_BASES will be represented by a dot (period). The bases corresponding to reads not uniquely aligned will all be represented in the same manner, as will any unknown bases in reference sequence to which reads have been uniquely aligned.

## Creating a Quality Table

In this section, the creation of a quality table (qtable file) is explained.

## Example

A simple example of quality table generation is :

```
Pipeline/QCalibration/QualityCalibration
    --cfg Pipeline/QCalibration/
    QualityCalibration.xml
    s_5_0001_qval.txt > s_5_0001_qtable.txt
```

- ▶ You can specify an arbitrary number of qval files.
- ▶ The qval file could have more than four parameters.



NOTE

The calibration routines are agnostic as to the number of parameters, so it would be easy to use a different set of parameters with a different extractor script. The binning for these parameters must be configured in an XML file such as `QualityCalibration.xml`. The path is the first argument expected by `QualityCalibration`. As yet, the implementation of `QualityCalibration` does not support the auto-generation of parameter binnings.

## Parameter Binning

Two schemata are supported for the specification of parameter bin boundaries.

- ▶ One allows arbitrary binnings to be supplied explicitly as a list.

```
<Parameter>
  <Name>Predictor</Name>
  <BinBoundaries>-0.7 0.3 4.2 20.5 101.0</
    BinBoundaries>
</Parameter>
```

- ▶ The other allows concise specification of a binning in which all bins except the end ones are of equal width.

```
<Parameter>
  <Name>Purity Score Per Base</Name>
  <FirstBinLowBound>-0.025</FirstBinLowBound>
  <FirstBinHighBound>0.525</FirstBinHighBound>
  <LastBinLowBound>0.975</LastBinLowBound>
  <LastBinHighBound>1.025</LastBinHighBound>
  <NumBins>11</NumBins>
</Parameter>
```

## Input Data

`QualityCalibration` derives the name of a corresponding `qref` file (containing the reference bases that it needs) from each `qual` file that is specified. It does this by replacing the `qual` filename suffix (by default, `_qual.txt`) with the `qref` filename suffix (by default, `_qref.txt`). These suffix strings can be overridden if required (see *Overriding Suffix Strings* on page 109).

The `qual` files can either be listed as explicit arguments to `QualityCalibration` or can be derived from the contents of a tile list file, `tiles.txt`. In the latter case, the `qual` file names are generated from those tile names in the tile list file that match a specified prefix, by appending the default or specified `qual` file suffix.

## Output Data

In standard Pipeline practice, the prefix corresponds to a lane (e.g. "s\_1" for lane 1) and the output is written to one quality table for that lane for single read analysis.

## Paired-End Analysis

For paired-end analysis, two per-read quality tables are produced by separate applications of QualityCalibration to each per-read set of qval files, with specification of the latter by suffix strings that include the read number (e.g. `_1_qval.txt` could be used for read 1 and `_2_qval.txt` for read 2).

## Compression

QualityCalibration can read from compressed qval (and/or qref) files, if this is specified. The corresponding suffixes will then have to be specified in full as QualityCalibration does not automatically append compression-related extensions.

## Detailed Usage of QualityCalibration

The detailed usage of the QualityCalibration binary is as follows:

- ▶ For information about usage:

```
./QualityCalibration -h|--help
```

- ▶ For creating the quality table (with available options listed):

```
./QualityCalibration[-S|--qval_suffix _qval.txt]
[-s|--qref_suffix _qref.txt]
[-Z|--qval_compression none|gzip|bzip2]
[-z|--qref_compression none|gzip|bzip2]
-c|--cfg <cfg_file.xml>
< <qvalue-file1> [<qvalue-file2> ...]
| -T|--tile_list_file <tiles.txt>
-t|--tile_prefix <lane_prefix e.g. s_3> >
```

## Overriding Suffix Strings

The `qval_suffix` and `qref_suffix` options (with default values as shown above) are used to generate qref filenames from the specified qval filenames. In addition, if the qval files are specified by means of a tile list file plus lane prefix filter, the `qval_suffix` is appended to the tile names to generate the qval filenames. In this case, non-default suffix values may be required (e.g. `_1_qval.txt` and `_1_qref.txt` for read 1 in eland pair analysis).

## Generating New Quality Values

In standard Pipeline practice, QualityApply is run once per lane (or per read per lane) to produce a per-lane (or per-read, per-lane) qcal file from all the corresponding qval files.

## Example

The following is a simple example of reestimation of quality values:

```
Pipeline/QCalibration/QualityApply
--orig_read_lengths 36:36
s_5_qtable.txt s_5_0001_qval.txt \
> s_5_0001_qcal.txt
```

This shows the reestimation of the quality values for base-calls within the reads in one tile, based upon a quality table calculated for the whole lane.

## Input Data

As with QualityCalibration, multiple qval files may be specified by means of a tile list file as an alternative to listing them explicitly.

Unlike QualityCalibration, QualityApply does not use qref files.

## Output Data

The default output format for the reestimated quality values is symbolic. Specifically, the value is represented by the ASCII character for which the code is the value added to 64, allowing a range of negative as well as positive values to be compactly displayed. Each output row is a single string of such characters, corresponding to the bases of a single read.

Numeric output may be specified instead; each output row again corresponds to a single read but consists of space-separated numbers.

## Compression

Once again, the qval files may be supplied and used in compressed form if this is specified; the use of any compression extension will require the explicit specification of a qval file suffix including it.

## Detailed Usage of QualityApply

The detailed usage of the QualityApply binary is as follows:

- ▶ For information about usage:  

```
./QualityApply -h|--help
```
- ▶ For generating new quality values (with available options listed):

```
./QualityApply
-l|--orig_read_lengths <ORIG_READ_LENGTHS>
[-r|--read READ_NUM]
[-Z|--qval_compression none|gzip|bzip2]
[-n|--numeric | -s|--symbolic]
<qtable-file>
< <qvalue-file1> [<qvalue-file2> ...]
| -T|--tile_list_file <tiles.txt>
  -t|--tile_prefix <lane_prefix e.g. s_3>
[-S|--qval_suffix _qval.txt] >
```



### NOTE

If the qval files are specified by means of a tile list file plus lane prefix filter, the qval\_suffix is appended to the tile names to generate the qval filenames.

## Paired-End Analysis

A non-default suffix value may be required for paired-end analysis (e.g. `_1_qval.txt` for read 1 and `_2_qval.txt` for read 2).

## Configuring Quality Table Sources in GERALD

By default, the Pipeline generates a quality table for each lane in which an analysis including alignment is performed. Then, Pipeline uses this quality table (or pair of quality tables) to reestimate the base-call quality values of all the tiles in that lane.

The source of the quality table(s) used in the quality calibration for a lane may be overridden by defining QCAL\_SOURCE (or, for individual reads within a paired-end analysis, QCAL\_SOURCE1 and/or QCAL\_SOURCE2) in the config.txt passed to GERALD.pl to configure GERALD analysis.

### Supported Values of QCAL\_SOURCE Variables

Supported Value	Description
auto	the qtable(s) used within the lane (to re-estimate the base-call quality values) will be the qtable(s) generated for that lane (from the quality predictor values and called and reference base values of bases in reads from that lane)
auto<n>, where n is the number of a lane for which alignment will be performed	the qtable(s) used in the lane will be those generated for lane n, e.g. `auto5' means that the qtable(s) from lane 5 will be used
/path/to/qtable.txt	the qtable file at the specified path will be used

As with any config.txt variable, the QCAL\_SOURCE variables may be specified on a flow cell-wide basis or for any non-overlapping subsets of the flow cell lanes, with the latter overriding the former if both are specified.

### Paired-End Analysis

In a paired-end analysis lane, specification of QCAL\_SOURCE1 and/or QCAL\_SOURCE2 will override specification of QCAL\_SOURCE, although the latter will be used if it has been specified and not overridden for a given read.

The interpretations for cases where paired-end analysis is in use for either or both of the source lane and the target lane are intended to be based upon the principle of least surprise:

1. If both lanes are paired, then any specification of the source lane for read 1 of the target lane results in the read 1 qtable of the source lane being used as the read 1 qtable in the target lane - and similarly for read 2.
2. If only the target lane is paired, then there is only one qtable available in the source lane but it may be used for both reads in the target lane.
3. If only the source lane is paired, then the read 1 qtable is used.

### Example

```
ANALYSIS eland_pair
QCAL_SOURCE /home/illumina/ref42_qtable.txt
123:QCAL_SOURCE auto8
4:QCAL_SOURCE1 /home/illumina/ref51_qtable.txt
4:QCAL_SOURCE2 auto7
```

The previous example specifies the following qtable usage:

1. Lanes 1–3 read 1 will use the lane 8 read 1 qtable; lanes 1–3 read 2 will use the lane 8 read 2 qtable
2. Lane 4 read 1 will use the external ref51\_qtable.txt qtable
3. Lane 4 read 2 will use the lane 7 read 2 qtable
4. Lanes 5–8 read 1 and read 2 will use the ref42\_qtable.txt qtable

Even though the lane 8 qtables will not be needed in lane 8, they will still be generated for use in lanes 1–3.

## Default and Experimental Predictors

The `extract_quality_predictors` script produces four default base-call quality predictors. In addition, three experimental predictors are currently supported for development purposes. All of these predictors are listed in this section.



NOTE

Future releases may contain additional experimental predictors.

### Default Base-Call Quality Predictors

The default base-call quality predictors produced by `extract_quality_predictors` are listed in Table 31 on page 112.

**Table 31** Default Base-Call Quality Predictors

Predictor	Description
<code>in_read_cycle</code>	This is the (1-offset) cycle number; for multiple read analysis this cycle number is relative to the read in which the base occurs, e.g. it would be 2 rather than 38 for the second cycle of the second read of a pair of 36-cycle reads
<code>unchastity</code>	The 'chastity' statistic for a base is defined as the ratio of the highest of the four (base type) intensities to the sum of highest two; the 'unchastity' predictor is simply the chastity value subtracted from 1.
<code>max_early_unchastity</code>	This is the maximum unchastity value over the first PURE_BASES (12 by default) bases in the read in which the base of interest occurs.
<code>raw_unquality</code>	This is the negative of the highest of the base type probabilities estimated by Bustard for the base.

### Experimental Base-Call Quality Predictors

The experimental predictors for development purposes are listed in Table 32 on page 113.

**Table 32** Experimental Base-Call Quality Predictors

Predictor	Description
max_local_unc hastity	This is the maximum unchastity over a window of bases centred on the base of interest; two neighbours on either side are considered.
homopol_len	This is the length of the run of the same called base type in which the base of interest occurs; the "run" will be of length one if the base type called for the base of interest differs from both that called for the base before it and that called for the base after it in the read. (This initial predictor ignores the position of the base of interest within a homopolymer run.)
signal_decay	This is the proportion by which the highest intensity associated with the current base is less than the highest intensity associated with the first cycle base. The value is constrained by thresholding to be in the intuitively expected range 0 to 1; possibilities such as a highest intensity increase or negative intensities (these can result from scaling of the intensities by Bustard) might otherwise result in a value outside this range.

## Further Considerations

Take the following points into consideration when performing your calibration:

- ▶ Best achievable Q score depends on total number of data points (because of the  $1 + \dots$  term in the version of the phred formula implemented in the paper).
- ▶ The quality of alignments in training data set is also crucial.
- ▶ The parameters used are not always strictly monotonic predictors for data quality. Furthermore, there is a tendency of the phred algorithm to overfit and there are cases when subsets of the data actually have higher quality scores than earlier rules.

## Frequently Asked Questions

### What's the difference between Illumina's base scores and Phred scores?

Like Phred scores, the Illumina base scoring scheme is just a way of expressing estimates of sequencing error probability in a convenient form.

Many people are familiar with Phred scores, named after the Phred base-calling software developed by Phil Green and coworkers. A Phred score of a base is

$$Q_{\text{phred}} = -10 \log_{10}(e)$$

where  $e$  is the estimated probability of a base being wrong. If a base is estimated to have a 1% chance of being wrong, it gets a Phred score of 20. Phred score=30 corresponds to 0.1% estimated error, 40 to 0.01%, and so on.

The base scores produced by the base caller use a 4-values-per-base scheme which also encodes information on the next most likely base call. After some discussion with James Bonfield at Sanger, the following score was chosen:

$$Q_{\text{Illumina}} = 10 \log_{10}(p(X)/(1-p(X)))$$

You will get one positive score - that's the score of your base call - and three negative scores.

To convert from a Illumina score back to a probability value use:

$$p(X) = 1 - 1/(1+10^{(Q_{\text{Illumina}}/10)})$$



#### NOTE

Important point: the highest Illumina base score and the Phred score are asymptotically identical. This means that for scores of about 15 and above they are so close as to be effectively the same.

For the precise of mind the exact formula is:

$$Q_{\text{Illumina}} = Q_{\text{phred}} + 10 \log_{10}(1-e)$$

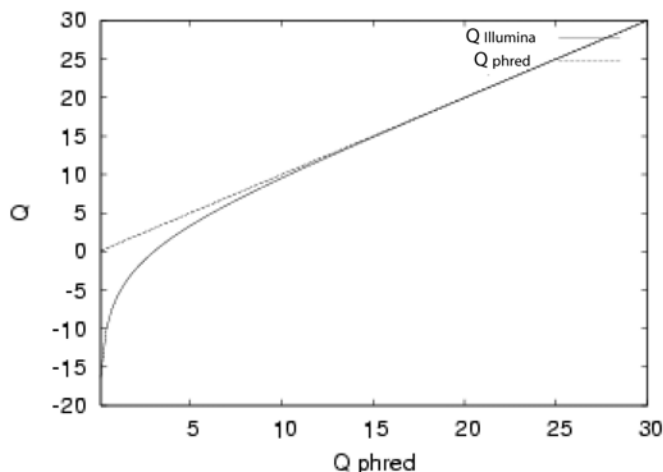


Figure 6  $Q$  vs.  $Q_{\text{phred}}$



It's important to separate the notion of a base scoring scheme from the base scores themselves, which are just estimates of error probability as encoded by the scheme. The Bustard base caller produces error rates in the 4 values per base format, these are held in the `_prb.txt` files in the Bustard directory.

### What are the different quality scores used by the PhageAlign program?

When run in the "ANALYSIS default" mode, the Pipeline uses the PhageAlign program to align reads against a reference, weighting each base according to the intensity-based quality values produced by the base caller. The output of this goes in the `_qalign.txt` files. The quality scores are not calibrated, underestimate base quality for lower quality values, and overestimate base quality for higher values. However in "ANALYSIS default" mode the Pipeline also generates its own "empirical quality values" and realigns using those values. This predates intensity-based quality values being produced by the base caller and still provides a useful validation of the intensity-based quality values.



At the moment PhageAlign still uses the scoring scheme  $S=100\log_{10}(p(X)/(1-p(X)))$ , i.e. 10 times as high as the `_prb.txt` scores. This will be fixed in the next Pipeline release.

This is done as follows:

A first PhageAlign alignment is done giving all bases equal weight and stored in the `_align.txt` files. A match base is arbitrarily given a score of 500, equating to 100000:1 chance of the base being wrong. If the base is assumed to have an equal probability of being mis-sequenced as each of the three possible mismatches, a score of -547 for a mismatch results.

This alignment is used to compute an empirical error probability for each base at each cycle.

These probabilities are converted to Illumina scores and stored in the `_score.txt` files.

The sequences are realigned, weighting the bases according to the alignment scores, and stored in the `_prealign.txt` files. The alignments are purity filtered and the results go in the `_realign.txt` files. See *Intermediate Output Data Files* on page 83 for file descriptions.

### How valid is it to estimate base quality by looking at alignments?

Depends on:

- What you are aligning against.
- What you are using to do the alignment.

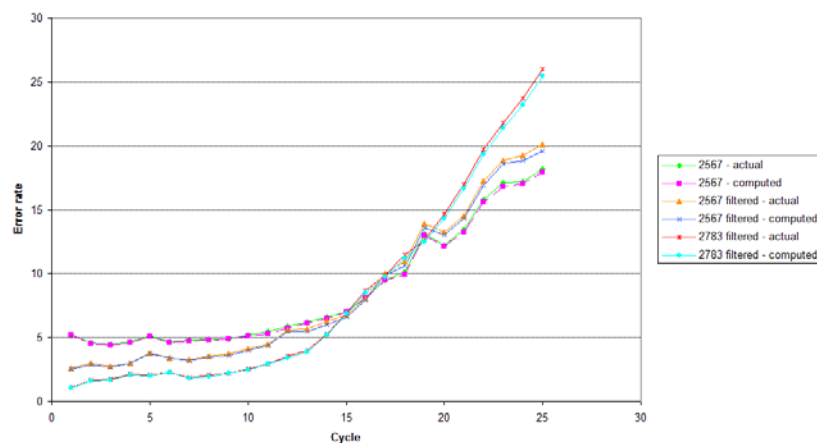
When estimating error rates from alignments, keep in mind that what you want is the probability that a base is wrong. However, what you actually get is the probability that a base is wrong given that the read it is in is uniquely alignable to your reference. This in turn depends on several things:

- a. How sensitive your alignment program is. The ELAND program only detects alignments with at most two errors per fragment, therefore the noisier reads having three or more errors will be ignored, meaning that error rate estimates obtained from ELAND alignments underestimate the true error rate.
- b. The uniqueness of reads in your target. This in turn depends on your read length, the length of your reference sequence, and how repetitive your reference sequence is. Obviously, only the first of these is (mostly) under your control. If your read length is too short to compensate for the other two factors, then it may be the case that a read with three errors will often also be “three errors similar” to another position in the reference, as well as to its originating position. It therefore can not be uniquely aligned.

This means, again, that noisier reads tend to be “lost” and so not contribute to the error rate calculation. So, again, the estimated error rate will underestimate the true error rate, but it’s important to realize that we get the same effect as in point a, but for a different reason. Even if we have a “perfect” alignment program, we are still subject to this phenomenon.

- c. Contaminants in the sample may contribute to error rates depending on the read-length, target sample, and the edit distance of the contaminant reads to the target genome.

The PhiX or human BAC sequencing runs performed for validation purposes mitigate both point a and b to a large extent, meaning that the calculated error rates closely estimate the true error rate. The PhageAlign program allows any number of errors per fragment, and reads of the default read length of 25 are able to be uniquely aligned to the BAC even in the presence of six or seven sequencing errors in a single fragment.



**Figure 7** Actual vs. Computed Error Rate for Three Sets of Simulated Reads

Figure 7 shows the effect of generating simulated BAC reads whose distribution of sequencing errors matched that observed in three actual experiments. An error rate was then computed by analyzing the simulated reads with the Pipeline. There is an extremely close match between the “actual” and “computed” error rates (and also

that the computed error rates slightly underestimate the actual error rates, for the reasons outlined above).

### Where are the alignment scores in the PhageAlign output?

```
AGTAGGAGGTGAGGCGGGGAGTAGG 5171 1 150002 F  
AGCAGCAGCAGAAGCAGGGAGGAGG 4124
```

Field 1 is the read, field 2 is the score of its best alignment. Field 3 is the number of positions in the reference sequence to which the read aligns with that score. If this is 1 (i.e. there is a unique best match), then a further four fields are present. Fields 4 and 5 are the position and strand of the unique best match, field 6 is the portion of the reference the read aligned to, and field 7 is the score of the next highest match.

### Where are the alignment scores in ELAND output?

Alignment scores are produced as part of both `eland_extended` and `eland_pair` analyses and the results are placed in the export file. See Table 28, *Final Output File Formats*, on page 86 for a description of the export file.

### How do Illumina scores relate to BLAST scores?

Short answer: not closely. Most of the existing work on alignment statistics, including Karlin-Altschul statistics and the theory behind BLAST, pertains to local alignments: given two sequences, e.g. your 500 base pair read and a public database, find the "maximal scoring pair" where subregions of the two sequences having the highest alignment score according to some scoring scheme. This problem is exactly solved by the Smith-Waterman algorithm and approximately solved more quickly by BLAST and other alignment programs.

The alignment problem for Illumina and other short reads is a global alignment problem: what is the best alignment of all bases of the read in the target database?



# Index

## A

- All.htm file 57
- analysis modules 6
- analysis output 2, 48
  - cluster intensity 81
  - error rates 57
  - file formats 86
  - file naming 58
  - lane averages 56
  - proportion of reads 57
  - quality scores 82
  - raw sequences 81
  - summary 48
  - tile-by-tile 57
- analysis parameters 30
- ANALYSIS variables 29

## B

- base calling 2
- Bustard 7
- bustard.py script 6, 22, 66

## C

- clean 24
- clusters passing filters 62, 63
- clusters per tile 61
- compression 22, 25
- config.txt file 21, 33, 35
- contaminant filtering 33
- control lanes 22, 66
- cycle selection 21

## D

- data folder 10
- default\_offsets.txt 13

## E

- ELAND 7, 16, 41
  - analysis modes 29

- error rates 57
- eland\_extended 29, 42
- ELAND\_MAX\_MATCHES 43
- ELAND\_MULTIPLE\_INSTANCES 40
- eland\_pair 29, 43
- ELAND\_SEED\_LENGTH 43
- ELAND\_standalone.py script 68
- eland\_tag 29, 41
- ELAND\_test.pl script 70
- email reporting 75
- error rates 16, 57, 63
- Error.htm file 57

## F

- fasta file format 38, 41
- filtering parameters 31
- Firecrest 7
- first cycle intensity 61
- FORCE option 33
- frequency cross-talk 7, 14, 22

## G

- gene expression 29
  - eland\_tag 41
- GERALD 7, 28
- GERALD.pl script 6, 28, 67
- GOAT 6
- goat\_pipeline.py script 6, 19, 22

## H

- hardware requirements 73
- help
  - reporting problems 4
  - technical support 4

## I

- image analysis 2
- images folder 10
- installation 77
- intensity curves 64

IPAR analysis 23  
IPAR folder 20  
IVC.htm file 56

## L

lane selection 36  
Linux Red Hat 75

## M

make 6, 21, 94  
make recursive 19, 25  
makefile targets 24  
matrix file 22  
matrix.txt file 14  
monotemplate 29

## N

network requirements 72  
new-read-cycle 12, 21  
nobasecall 22  
nohup 20

## P

PAIR\_PARAMS 44  
paired reads 12

- analysis variables 29
- command line variations 23
- configuration file options 37
- eland\_pair 43

parallelization 20, 24, 94

- limitations 97

parameters file 12

- Data folder 10
- file format 89
- image analysis folder 11

parameters files

- contents 6

Perfect.htm file 57  
PhageAlign 7, 16

- error rates 57

phasing 7, 22, 23  
phasing.xml file 15  
phasing/prephasing percentage 63  
prephasing 7, 23

## Q

quality filtering 33  
quality scores 82, 100

## R

Run Folder 8

- naming 11
- structure 9

run quality 60  
runReport.pl script 75

## S

sequence alignments 2, 28  
sequence\_pair 30  
software requirements 75  
squashGenome 38  
standard deviations 63  
Summary.htm file 48, 60

## T

technical support 4  
tile selection 21, 24, 67  
tile variability 64

## U

USE\_BASES 30, 31



Illumina, Inc.  
9885 Towne Centre Drive  
San Diego, CA 92121-1975  
+1.800.809.ILMN (4566)  
+1.858.202.4566 (outside North America)  
techsupport@illumina.com  
[www.illumina.com](http://www.illumina.com)

