illumina

Off-Line Basecaller v1.9 User Guide

FOR RESEARCH USE ONLY

ILLUMINA PROPRIETARY
Part # 15009920 Rev. C
November 2010

Revision History

Part #	Revision	Date	Description of Change
15009920	C	November 2010	Supports in-line sample prep controls; separate pre-determined calibration tables for Genome Analyzer and HiSeq data; deprecated image analysis; base calling generates BCL files by default; updated base calling algorithms.
15009920	B	June 2010	Contains the BCL Converter; updated image analysis and base calling algorithms.
15009920	A	November 2009	Launch

Table of Contents

List of Tables

Overview

Chapter 1

Introduction

This user guide documents the Off-Line Basecaller (OLB), which performs base calling and bcl to qseq conversion for the HiSeq or Genome Analyzer.

The standard workflow is to perform base calling using on-instrument real time analysis (RTA), after which CASAVA performs alignment using the base calling results. If needed, OLB provides the option to perform data analysis off-line. In addition, OLB allows you to convert per-cycle base call files (*.bcl) into per read base call files (*_qseq.txt).

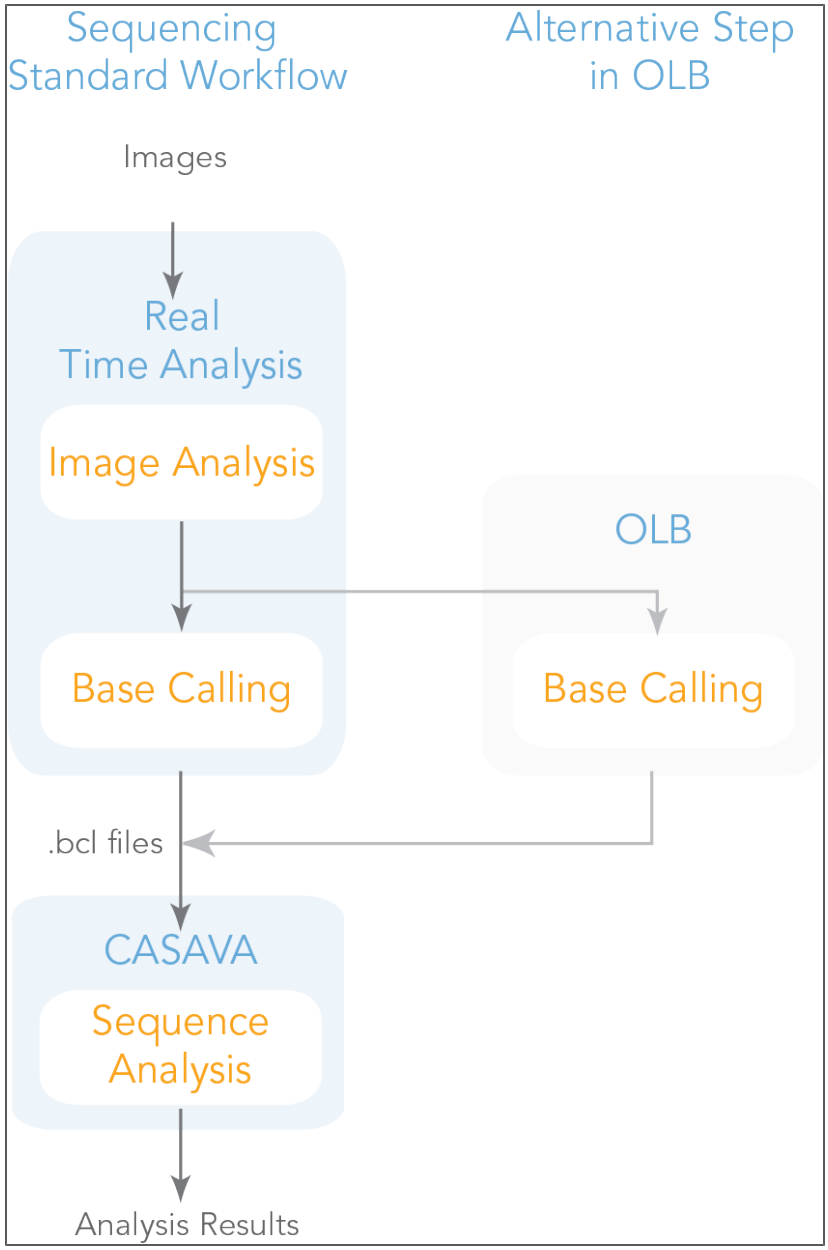
The basic functionalities of the modules in OLB are described below.

Analysis of Sequencing Data

After the sequencing platform generates the sequencing images, the data is analyzed in two steps: image analysis and base calling. CASAVA then uses the sequencing output to align the reads to a genome, call SNPs, detect indels, and count reads (for RNA sequencing).

Base calling uses cluster intensities and noise estimates to output the sequence of bases read from each cluster, a confidence level for each base, and whether the read passes filtering. Base calling is performed by RTA by default, but can be done off-line by OLB.

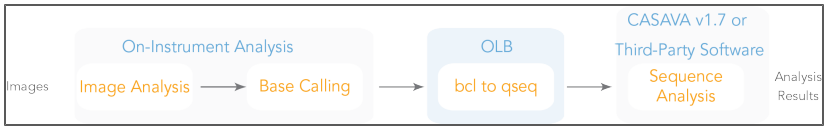
Figure 1 OLB Data Analysis Steps



BCL Conversion

The standard sequencing output for the HiSeq2000 consists of *.bcl files, which contain the base calls and quality scores per cycle. CASAVA v1.8 can use *.bcl files as input, but older versions of CASAVA (v1.7 and older), as well as some third-party software, need *_qseq.txt files. To convert *.bcl files into *_qseq.txt files, use the BCL Converter in OLB.

Figure 2 HISeq2000 Data Analysis Standard Workflow



NOTE

The BCL conversion should be started only after completion of the run. RTA updates the config.xml on every cycle and this interferes with the configuration of the BCL Converter. Running the BCL Converter while RTA is still updating the BaseCalls directory will lead to errors that can be difficult to detect, identify and diagnose.

For intermediate BCL conversions (before completion of the run), Illumina recommends to do the base calling offline, from the CIF files, using 'bustard.py'.

Analysis Computing Systems

Base calling can either be performed in real time or off-line by two different analysis computing systems:

- Real time analysis (RTA), which runs on the sequencing platform instrument computer. RTA performs real-time image analysis and base calling.
- The Off-Line Basecaller (OLB), which runs on a Linux analysis server.

Bcl to qseq conversion should be done using OLB.

NOTE

OLB v1.9 contains the Bustard base calling module which uses the same algorithms as RTA v1.9 and v1.10.

Run Folder

The output data produced by the Off-Line Basecaller are stored in a hierarchical folder structure called the Run Folder. The Run Folder includes all data folders generated from the sequencing platform and the data analysis software. For a detailed description of the Run Folder structure, see *Understanding the Run Folder* on page 12.

OLB requires a Linux system with specific processing and data storage capacity. For specific requirements, see *System Requirements* on page 1.

Installation

1. Install the OLB prerequisites on a suitable Linux system. See *Installation Prerequisites* on page 48.
2. Install the OLB software and compile OLB using the “make” command. See *Installing the OLB Software* on page 50.
3. Set up the “Instruments” directory for parameters files. See *Directory Setup* on page 50.

What's New

Important Changes in OLB v1.9

- Supports in-line sample prep controls.
- Generates ControlsReport.csv in-line controls output file.
- Separate pre-determined calibration tables for Genome Analyzer and HiSeq data; OLB automatically detects which one to use.
- Image analysis has been deprecated.
- Base calling generates BCL files by default.
- Supports formatted (_pos.txt), binary (.locs), or compressed binary (.clocs) position files.
- Updated base calling algorithms, conforming to RTA v1.9 and v1.10.

Important Changes in OLB v1.8

- OLB v1.8 contains the BCL Converter.
- OLB v1.8 has updated image analysis and base calling algorithms, as used by RTA v1.8.

Important Changes in OLB v1.6

- OLB contains Firecrest image analysis and Bustard base calling modules that use the same algorithms as RTA v1.6.

Reporting Problems

Contact Illumina Technical Support to report any issues with OLB.

When reporting an issue, it is critical 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 *Nohup Command* on page 22. It helps to attach the makefile corresponding to the part of OLB that is causing the problem.

Core Data Analysis Concepts

Chapter 2

Introduction

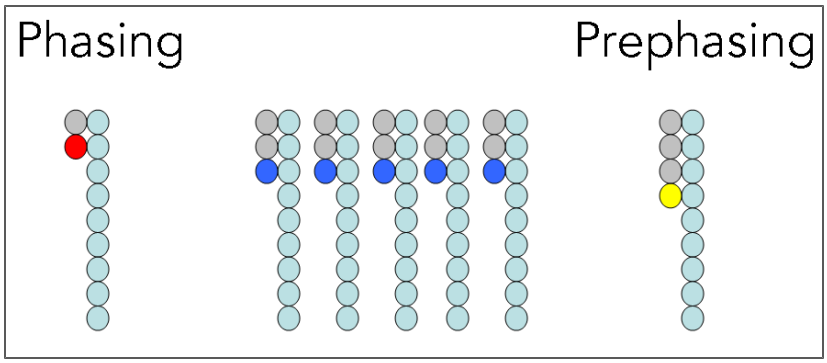
The main module of OLB performs off-line basecalling. During an analysis run, a defined folder structure is generated that captures the output of an instrument run in text files and also contains the configuration files. Configuration files contain calibration and input settings that optimize your analysis run. This chapter describes these core concepts of the Off-Line Basecaller.

Base Calling Module

OLB contains the Bustard for base calling. Bustard deconvolves the signal from the clusters and applies correction for cross-talk, phasing, and prephasing.

- Frequency cross-talk—The sequencing platform uses two lasers and four filters to detect four dyes attached to the four types of nucleotide, respectively. The emission spectra of these four dyes overlap so that the four images are not independent. OLB uses a frequency cross-talk matrix to correct for this cross-talk (for more information, see *Frequency Cross-Talk Matrix* on page 16).
- Phasing/Prephasing—Depending on the efficiency of the fluidics and chemistry of the sequencing reactions, a small number of molecules in each cluster may run ahead of (prephasing) or fall behind (phasing) the current incorporation cycle. This effect is mitigated by applying corrections during the base calling step (for more information, see *Phasing/Prephasing Estimates* on page 17).

Figure 3 Phasing and Prephasing



Use of Module

For base calling starting with image analysis data generated by RTA, use the bustard.py script. *Using Bustard* on page 19 describes the use of bustard.py.

Running the OLB Modules

OLB 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. Each OLB module is a collection of Perl or Python scripts and C++ executables, and has its own makefile associated with the analysis task.

"Make" has a dual purpose within the OLB software:

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

A run of OLB is a two-stage process:

1. Generate the folders and makefiles using one of the above scripts.
2. Start OLB analysis by executing "make."

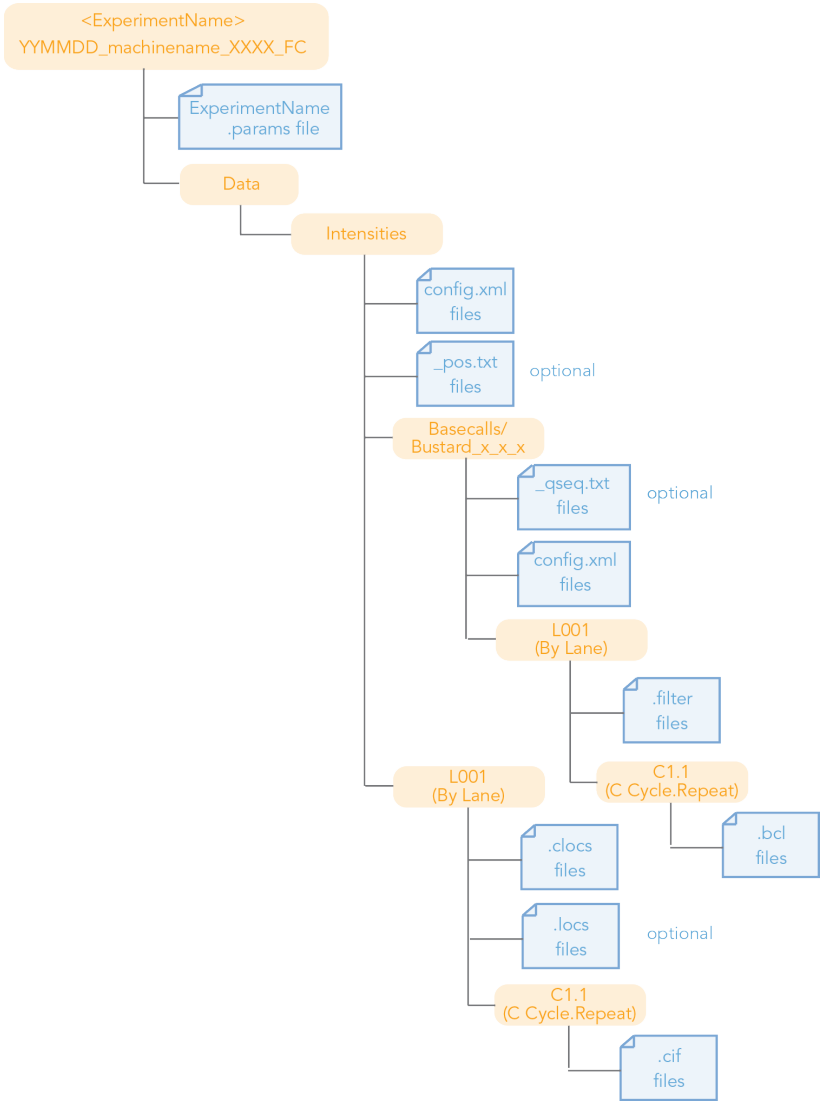
The process is described in *Using Bustard* on page 19.

Understanding the Run Folder

OLB operates in a specific directory called the Run Folder where the analysis output files are saved by default in a consistent hierarchical structure. A Run Folder containing RTA data is very similar to a Run Folder containing only OLB analysis data.

The figure below illustrates a typical Run Folder after image analysis and base calling.

Figure 4 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 flawlessly, 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 Foldercontains the Data folder as illustrated in *Understanding the Run Folder* on page 12 above. The Data folder contains Intensities folders and the Intensities folders contain Basecall folders which will contain the GERALD folder once alignment is started. The Data folder is created by the sequencing platform when a run starts. Any analysis performed on the data, including RTA, is saved within the Data folder.

Each run of the main OLB analysis modules creates a subdirectory in the Data folder of the Run Folder as follows (see *Understanding the Run Folder* on page 12 above):

- Each run of the OLB 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 for folders generated by OLB consists of the number of cycles run, the version of the software used for the operation (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.

Data Folder

The Data folder contains a hierarchical structure that consists of the image analysis output folder (Intensities), then the base calling output folder (Basecalls or Bustard_x_x_x), and then the sequence alignment output folder where CASAVA will output alignment results (GERALD).

A new subfolder is generated when a set of images is processed by RTA. The data are kept in one file per tile for raw intensities. RTA reports image analysis results in the binary .cif format (intensities).

The Data folder contains one config.xml file in each image analysis folder generated as a result of analyzing sets of images.The config.xml file explicitly records which cycle-image folders were used to generate the raw intensities and noise files, and any parameters used. For a detailed description of the parameters file, see *Configuration/Parameters* on page 15.

Base Calling Folders

Each image analysis folder may hold multiple sequence folders with the output of different runs of a base caller package. The base calling folders have the following naming structure:

- The base calling folder generated by RTA is called BaseCalls.

- Each base calling folder generated by Bustard is named using the following convention:
  <analysis module><analysis module-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 base calling folder also holds a config.xml that records any relevant information about the run of the base caller module.

Run Folder Naming

The top level Run Folder name is generated using three fields to identify the <ExperimentName>, separated by underscores. For example, YYMMDD_machinename_NNNN. You should not deviate from the Run Folder naming convention, as this may cause OLB to stop.

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.

NOTE
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.

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.

NOTE
When publishing the data to a public database, it is desirable to extend the exclusivity globally, for instance by prefixing each machine with the identity of the sequencing center.

File Naming

OLB uses the following format for file naming:

Table 1 File Naming Conventions

File	Description	Naming Convention
*.bcl	Base call and quality score file	<sample>_<lane>_<tile>.bcl
*.stats	File containing base calling statistics	<sample>_<lane>_<tile>.stats
*.filter	File containing filter results	<sample>_<lane>_<read>_<tile>.filter
*_pos.txt	Positions file	<sample>_<lane>_<tile>_pos.txt
*.locs	Template locations file	<sample>_<lane>_<tile>.locs
*.clocs	Compressed version of locs file	<sample>_<lane>_<tile>.clocs
*.cif	Cluster intensity file	<sample>_<lane>_<tile>.cif
*_qseq.txt	File containing base calls and quality scores per read	<sample>_<lane>_<read>_<tile>_qseq.txt

When a given file type is split on a read basis, the read always appears in the name, even for single-read analysis. Example: s_5_1_0030_qseq.txt is a valid filename.

Table 2 File Naming Components

Component	Description
<sample>	Alphanumeric string (always "s")
<lane>	Single-digit number identifying a flow cell lane
<read>	Single-digit number identifying the read (starts at 1)
<tile>	Four-digit number identifying a tile location in a flow cell lane
<cycle>	Two- or three-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

Configuration/Parameters

The Data Folder and subfolders can all contain a configuration file (config.xml), and the top level Run Folder a related .params file. This 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 *Configuration/Parameters File Format* on page 32.

Calibration and Input Parameters

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

Quality Scoring

Base quality value calibration now uses a pre-determined calibration table in Bustard, supplied with the software. There are separate quality tables for Genome Analyzer and HiSeq data, and OLB automatically detects which one to use.

The quality scoring scheme is the Phred scoring scheme, encoded as an ASCII character by adding 64 to the Phred value. A Phred score of a base is:

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

where e is the estimated probability of a base being wrong.

NOTE
You can always check whether bases have been called with the default GA qtable (no switch) or using --quality-table /path/to/HiSeqTable.txt by looking at the log files for the run.

Frequency Cross-Talk Matrix

The sequencing platform uses two different lasers to excite the dye attached to each nucleotide. The emission spectra of these four dyes overlap, 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.

As of OLB v1.9, four cycles are used for cross-talk matrix estimation. The advantage to using multiple cycles for matrix estimation is that it is more likely to get correct estimation even for less diversified samples, the matrix estimation is more robust.

The frequency cross-talk is estimated during the base calling run and captured in a file called s_<cycle>_matrix.txt. The s_<cycle>_matrix.txt file is located in the base calling folder in the Matrix subfolder.

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 C, A, T, and G dyes/nucleotides (columns) cross-talk into the C, A, T, and G 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 within the base calling directory.

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.

Using Bustard

Chapter 3

Introduction

This section describes the typical analysis run and command line options for Bustard. Use Bustard when you want to perform OLB analysis starting with the image analysis data.

The intensity data should be organized within a standard Run Folder directory structure as described in *Run Folder Structure* on page 13. To successfully initiate base calling, you need intensity and position files for every lane and cycle, and a configuration (config.xml) file in the Run Folder.

Invoking Bustard for Base Calling

Although several different software programs are involved in an analysis run, a single command generates the analysis folders, then a second command (`make all') can be used to start a complete analysis.

Below is the standard invocation of OLB when starting with image analysis data, for which the Bustard.py script needs to invoked. Arguments contained in brackets [ ] are optional.

```
/path-to-olb/bin/bustard.py  <image-analysis-directory>
--CIF [--bin-controls] [--no-controls] [--control-lane=5]
[--make] [--matrix=mymatrix.txt|auto|auto<n>]
[--phasing=0.01|auto|auto<n>] [--prephasing=0.01]
[--with-qseq] [--with-sig2] [--with-seq] [--with-qval]
[--GERALD=/path/config.txt] [--with-second-call]
```

Some of the arguments above have sample values displayed.

The only compulsory arguments is the path to the Run Folder that is to be analyzed. Generally, the --CIF argument is also required to indicate that analysis is started from *.cif files.

See *Command Line Options for Bustard* on page 23 for a detailed description of the options.

Running Off-Line Basecalling

Prerequisites Using Image Analysis Data

To process RTA data with OLB, you need the following:

- Experiment run folder containing the image analysis results folder must have been copied to the off-line server (for example /<RunFolder>/Data/Intensities)
- Config.xml files for the experiment have been copied to /<RunFolder>/Data/Intensities

Data Analysis

The RTA generated image analysis data can be analyzed in OLB in the following way:

1 Generate OLBmakefiles and analysis structure - this is done by invoking the bustard.py script :

```
/path-to-olb/bin/bustard.py --CIF <RunFolder>/Data/Intensities
   --make
```

All standard OLBparameters are available for use.

2 Execute the make files :

Navigate to the Bustard sub-directory generated in the Intensities directory. The "Makefile" for base calling generated in step one should be there. Do one of the following:

- To perform base calling only , execute the makefile in the Bustard directory using :

```
make all
```

Paired reads

The standard method to analyze paired-read data assumes that you have a single Run Folder containing the image analysis files for both reads, with a continuously incremented cycle count. OLB automatically knows where the second read starts.

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 in OLB* on page 51.

Nohup Command

You should 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 for Bustard

You can invoke the 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, OLB will not write any information to your Run Folder.

--new-read-cycle=<cycle>

Use this command to denote 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.

--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.

--compression=<method>

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

--GERALD=<config.txt>

Use this command to start the makefile generator for the GERALD alignment module in CASAVA. This happens after the Bustard folder is created, and passes the relevant analysis information to GERALD. 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 the CASAVA Software User Guide.

NOTE

For the --GERALD option to work, the bin directory of CASAVA has to be specified in the PATH environment variable. Type the following in your command line:

```
export PATH=path/to/CASAVA/bin
```

This path will now be valid for the current terminal session. See also your LINUX documentation for instructions.

Bustard Options

Use the following options with the bustard.py script.

--bin-controls, --no-controls

By default OLB will assume there are in-line controls, which control various sample preparation steps; see *In-Line Control Report* on page 29. OLB will write this information in the output files. There are two options you can use when there are no controls in the sample:

--bin-controls: this option tells OLB to assume there are controls, but they will be reported as reads that did not pass filter, so that no downstream analysis gets broken.

--no-controls: this option tells OLB that there are no controls in this sample.

--CIF

Indicates that analysis is started from *.cif files. Is required in most cases.

--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> | auto | auto<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," OLB 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. The value lane calculates a separate correction for each lane from data in that lane alone.

--matrix-cycles=n

The --matrix-cycles option specifies the number of cycles to be used for cross-talk matrix estimation.Default = 4.

--phasing=<x> | auto | auto<n> | lane

Use the --phasing command to apply a particular phasing correction. If you set the value to the default behavior "auto," OLB 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. The value lane calculates a separate correction for each lane from data in that lane alone.

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 prephasing correction. 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 OLB autogenerates phasing and pre-phasing estimates.

--with-qseq, --with-sig2, --with-seq, --with-qval

Use these commands to generate the qseq, sig2, seq, and qval files respectively. These files are not generated by default.

--with-second-call

When this flag is set, the second best base calls will be generated in the subdirectory SecondCall. The second best base calls are in sequence output files that mirror the original sequence output files, with an exact one-to-one match.

All information from the sequence output files in the SecondCall directory is exactly the same as that from the original sequence output files in the base calls directory, except for the sequence and quality scores:

- The second best base calls are based on the second highest value of the corrected intensities.
- The corresponding quality scores are set so that the sum of the probabilities of the actual base call and of the second best base call is equal to 1.

Paired Reads

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

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

Use either of these option formats to specify phasing 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 either of these option formats to specify the matrix for one specific read of a pair. They are analogous to the phasing options listed above.

Makefile Targets

The 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).

-j <n>

This parallelization switch can be used with the "make" command to execute the OLB run in parallel over <n> number of processor cores. For a description, see *Using Parallelization in OLB* on page 51.

clean

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

WARNING
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 analysis run all the way up to the 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 output 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.

Analysis Output File Descriptions

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

Output File Types

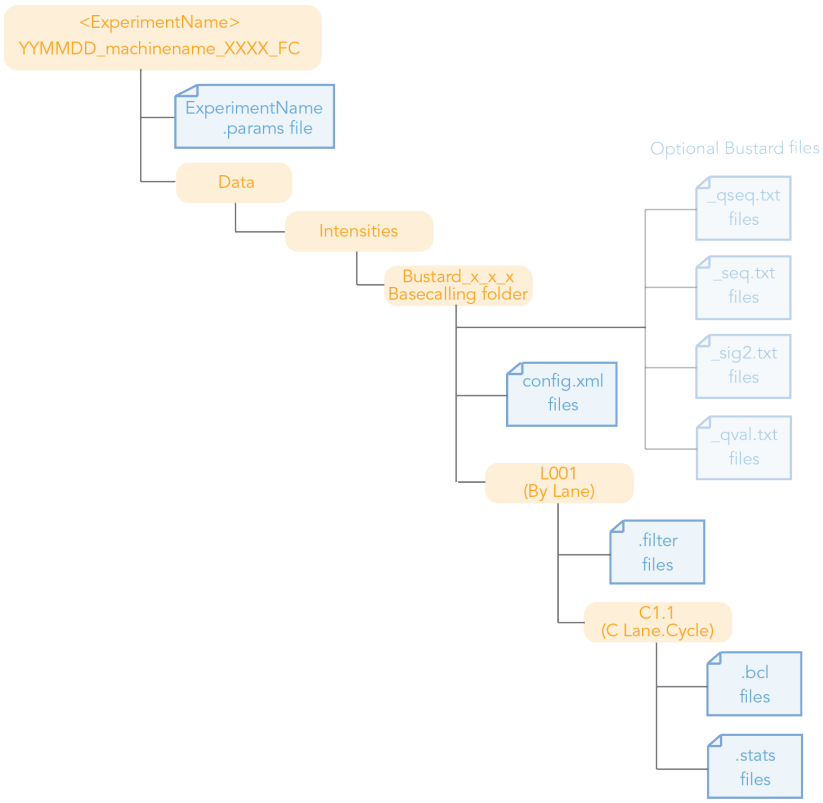
The main output files for Bustard are:

- *.bcl files, which contain the sequence data
- *.filter files, which contain the information whether the reads passed filter, and whether the read is an in-line control.
- *.stats files, which contain real time statistics for each cycle regarding associated *.bcl files, such as number of calls and average intensity.

In addition, Bustard can produce a number of optional files.

These files are described in this section; the location of these files is depicted in the image below.

Figure 5 Run Folder Structure and Output File Types



Bcl Files

The *.bcl files can be found in the BaseCalls directory:

```
<run directory>/Data/Intensities/BaseCalls/L<lane>/C<cycle>.1
```

They are named as follows:

```
s_<lane>_<tile>.bcl
```

The *.bcl files are binary base call files with the format described below.

Bytes	Description	Data type
Bytes 0–3	Number N of cluster	Unsigned 32bits little endian integer
Bytes 4–(N+3) Where N is the cluster index	Bits 0-1 are the bases, respectively [A, C, G, T] for [0, 1, 2, 3]: bits 2-7 are shifted by two bits and contain the quality score.	Unsigned 8bits integer

Stats Files

The stats files can be found in the BaseCalls directory:

```
<RunDirectory>/Data/Intensities/BaseCalls/L<lane>/C<cycle>.1
```

They are named as follows:

```
s_<lane>_<tile>.stats
```

The Stats file is a binary file containing base calling statistics; the content is described below. The data is for clusters passing filter only.

Start	Description	Data type
Byte 0	Cycle number	integer
Byte 4	Average Cycle Intensity	double
Byte 12	Average intensity for A over all clusters with intensity for A	double
Byte 20	Average intensity for C over all clusters with intensity for C	double
Byte 28	Average intensity for G over all clusters with intensity for G	double
Byte 36	Average intensity for T over all clusters with intensity for T	double
Byte 44	Average intensity for A over clusters with base call A	double
Byte 52	Average intensity for C over clusters with base call C	double
Byte 60	Average intensity for G over clusters with base call G	double
Byte 68	Average intensity for T over clusters with base call T	double
Byte 76	Number of clusters with base call A	integer
Byte 80	Number of clusters with base call C	integer
Byte 84	Number of clusters with base call G	integer
Byte 88	Number of clusters with base call T	integer
Byte 92	Number of clusters with base call X	integer
Byte 96	Number of clusters with intensity for A	integer
Byte 100	Number of clusters with intensity for C	integer
Byte 104	Number of clusters with intensity for G	integer
Byte 108	Number of clusters with intensity for T	integer

Filter Files

The filter files can be found in the BaseCalls directory:

```
<run directory>/Data/Intensities/BaseCalls/L<lane>/
```

They are named as follows:

```
s_<lane>_<read>_<tile>.filter
```

The *.filter files are binary files containing filter results; the format is described below.

Bytes	Description
Bytes 0–3	Zero value (for backwards compatibility)
Bytes 4–7	Filter format version number
Bytes 8–11	Number of clusters
Bytes 12–(2xN+11) Where N is the cluster index	The first byte is an unsigned char which indicates the identity of the control to which the read was matched, or is zero if the read wasn't identified as a control. The second byte has bit flags that are used as follows: • Bit 0 is pass or failed filter • Bit 1: Read was flagged as a control • Bit 2: Control match was ambiguous • Bit 3-7: Reserved for future use

In-Line Control Report

OLB reports the presence of in-line controls. The best way to view the control results is to open the report in the Sequencing Analysis Viewer (SAV). See the *Sequencing Analysis Viewer User Guide* for information.

If you want to analyze the raw data, use the ControlsReport.csv file. This file contains the columns Lane, Tile, Read, ControlName, Index, and Count.

Figure 6 ControlsReport.csv File Opened in Excel 2007

	A	B	C	D	E	F
1	Lane	Tile	Read	ControlName	Index	Count
23	1	5	1	CTA_250bp	GATCAG	0
24	1	5	1	CTA_250bp	GGCTAC	0
25	1	5	1	CTA_250bp	TAGCTT	0
26	1	5	1	CTA_350bp	ACAGTG	7
27	1	5	1	CTA_350bp	ATCACG	13
28	1	5	1	CTA_350bp	CGATGT	9
29	1	5	1	CTA_350bp	GCCAAT	11

export Sheet1

The in-line controls are added during the sample prep to check whether these steps are performed correctly, and OLB identifies them based on their sequence. The following controls are included:

Step	Reaction	Enzyme	Control
End-Repair	Chew-back of 3′ overhangs/ fill-in of 5′ overhangs	DNA Polymerase	CTE1
End-Repair	Phosphorylation of 5′ OH groups	Polynucleotide Kinase	CTE2
A Tailing	Add 3′ A overhang	DNA Polymerase	CTA
Adapter ligation	Join adapters to inserts	DNA Ligase	CTL

A successful run should have a positive signal for all four controls at one or two adjacent sizes. The reported size should correspond to the excised fragment. The absolute number of reads for each control does not matter much, but there should be a clear peak near the expected size.

If a certain control is not found in the sequencing sample, and it is the last control not found in that sample, it likely means that the corresponding reaction was inefficient. For example, if CTE1, CTE2 (End-repair controls), and CTA (A-tailing control) are absent, then probably the A tailing step went wrong, because that one happens after the end repair step.

Qseq Files

The _qseq files are not generated by default with the introduction of the *.bcl files, but can be saved using the switches `--with-qseq` as described in *Bustard Options* on page 24. They have the following format:

- Machine name: (hopefully) unique identifier of the sequencer.
- Run number: (hopefully) unique number to identify the run on the sequencer.
- Lane number: positive integer (currently 1-8).
- Tile number: positive integer.
- X: Xcoordinate of the spot. As of RTA v1.6, OLB v1.6, and CASAVA v1.6, the X and Y coordinates for each clusters are calculated in a way that makes sure the combination will be unique. The new coordinates are the old coordinates times 10, +1000, and then rounded.
- Y: Ycoordinate of the spot. As of RTA v1.6, OLB v1.6, and CASAVA v1.6, the X and Y coordinates for each clusters are calculated in a way that makes sure the combination will be unique. The new coordinates are the old coordinates times 10, +1000, and then rounded.
- Index: positive integer. No indexing should have a value of 0.
- Read Number: 1 for single reads; 1 or 2 for paired ends.
- Sequence
- Quality: the calibrated quality string.
- Filter: Did the read pass filtering? 0 - No, 1 - Yes.

NOTE

The quality scoring scheme Illumina uses is the Phred scoring scheme, encoded as an ASCII character by adding 64 to the Phred value. A Phred score of a base is:

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

where e is the estimated probability of a base being wrong.

Illumina's Read Segment Quality Control Metric

A number of factors can cause the quality of base calls to be low at the end of a read. For example, phasing artifacts can degrade signal quality in some reads, and the affected portions of these reads have high error rates and unreliable base calls. Typically, the increase in phasing causes quality scores to be low in these regions, and thus these unreliable bases are scored correctly.

However, the occurrence of phasing artifacts may not always correlate with segments of high miscall rates and biased base calls, and therefore these low quality segments are not always reliably detected by our current quality scoring methods. We therefore mark all reads that end in a segment of low quality, even though not all marked portions of reads will be equally error prone.

The read segment quality control metric identifies segments at the end of reads that may have low quality, and unreliable quality scores. If a read ends with a segment of mostly low quality (Q15 or below), then all of the quality values in the segment are replaced with a value of 2 (encoded as the letter B in Illumina's text-based encoding of quality scores). We flag these regions specifically because the initially assigned quality scores do not reliably predict the true sequencing error rate. This Q2 indicator does not predict a specific error rate, but rather indicates that a specific final portion of the read should not be used in further analyses.

This is not a read-level filter; the occurrence of consecutive Q2 values in a read does not indicate that the read itself is unreliable, but rather that only the base calls flagged with Q2 are unreliable. Note, however, that these regions are included in the Gerald error rate calculations for aligned reads. In typical sequencing runs, most reads are reliable over their entire length, and are not marked with Q2 indicators. Of the reads that are marked with the Q2 indicator, most are flagged only in the final few cycles. The number of reads marked by the quality control indicator, and the extent of the marking, can be used as an overall run quality metric.

Seq Files

The seq files are not generated by default with the introduction of the qseq files, but can be saved using the switches `--with-seq` as described in *Bustard Options* on page 24.

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 has not been applied.
- Signal purity filtering of low quality data has not been applied.
- There is one file per tile, resulting in 960 files in total for the GA$_{IIx.}$

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>
```

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. With real time analysis generated intensities the files are further broken down per cycle

Configuration/Parameters File Format

.Params File

The top level Run Folder contains a parameters file, named <Run FolderName>.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>
```

Config.xml Files

In the top level of the Data folder you will find the config.xml 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.x.x" />
<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="C1-24_Firecrest1.x.x_30-07-2007_user.2">
...
</Run>
</ImageAnalysis>
```

In each image analysis folder there is another config.xml 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>
```

RunInfo.xml File

The top level Run Folder contains a RunInfo.xml file. The file RunInfo.xml (normally generated by SCS) identifies the boundaries of the reads (including index reads).

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

```
<?xml version="1.0"?>
<RunInfo>
<Run Id="071112_SLXA-EAS1_0089_FC20120_R1" Number="89" >
<Instrument>SLXA-EAS1</Instrument>
<Reads>
<Read FirstCycle="1" LastCycle="30" />
<Read FirstCycle="31" LastCycle="37" >
<Index Name="xxx" FirstCycle="31" LastCycle="37" />
</Read>
</Reads>
<SecondEnd FirstCycle="38" />
<ActualIndex>
<Cycle>31</Cycle> <Cycle>32</Cycle>
    <Cycle>33</Cycle> <Cycle>34</Cycle>
    <Cycle>35</Cycle> <Cycle>36</Cycle>
</ActualIndex>
</Run>
</RunInfo>
```

Converting BCL Files

Chapter 5

Introduction

The standard sequencing output for the HiSeq™ and Genome Analyzer consists of *.bcl files, which contain the base calls and quality scores per cycle. Older versions of CASAVA (before v1.8) and third-party software use *_qseq.txt files as input. To convert *.bcl files into *_qseq.txt files, use the BCL Converter.

The BCL Converter script (setupBclToQseq.py) sets up the following steps:

- Conversion of the *.bcl files into *_qseq.txt files. This operation also requires *.filter files, and the following position files:
  - *.clocs files from a HiSeq (RTA 1.10) run
  - *.locs files from a GA (RTA 1.9) run
  - *_pos.txt files
- Conversion of the stats files into the corresponding `SignalMeans/s_<lane>_<tile>_all.txt files`.
- Completion of the building of the BaseCalls directory.

NOTE

The BCL conversion should be started only after completion of the run. RTA updates the config.xml on every cycle and this interferes with the configuration of the BCL Converter. Running the BCL Converter while RTA is still updating the BaseCalls directory will lead to errors that can be difficult to detect, identify and diagnose.

For intermediate BCL conversions (before completion of the run), Illumina recommends to do the base calling offline, from the CIF files, using 'bustard.py'.

Converter Input Files

The BCL Converter needs the following input files from RTA:

- *.stats files. For a description of this format, see *Stats Files* on page 28.
- *.filter files. For a description of this format, see *Filter Files* on page 28.
- *.bcl files. For a description of this format, see *Bcl Files* on page 27.
- *_pos.txt, *.locs, or *.clocs files
- config.xml file

RTA is configured to copy these files off the instrument computer machine to the BaseCalls directory on the analysis server. The files are decribed below.

Position Files

Locs files

The locs files can be found in the Intensities directory:

```
<run directory>/Data/Intensities/L<lane>
```

They are named as follows:

```
s_<lane>_<tile>.locs.
```

Clocs files

The clocs files are compressed versions of locs file and can be found in the Intensities directory:

```
<run directory>/Data/Intensities/L<lane>
```

They are named as follows:

```
s_<lane>_<tile>.clocs.
```

Pos Files

The *_pos.txt files can be found in the Intensities directory:

```
<run directory>/Data/Intensities
```

They are named as follows:

```
s_<lane>_<tile>_pos.txt
```

The *_pos.txt files are text files with 2 columns and a number of rows equal to the number of clusters. The first column is the X-coordinate and the second column is the Y-coordinate. Each line has a <cr><lf> at the end.

Config.xml File

The config.xml file can be found in the BaseCalls directory:

```
<run directory>/Data/Intensities/BaseCalls
```

The config.xml file contains contains meta-information about the run, image analysis, and base calling meta-information about the run, image analysis, and base calling.

Converter Usage

Typical uses of the BCL Converter are documented below.

Note that OLB detects which version of RTA was used for base calling, based on the position and format of the *.filter files. OLB then uses the filter file and positions file locations and formats that were generated by default for that RTA version.

If OLB cannot find the proper files it will state so in an error message, and you can specify the non-standard locations and formats by using the `-f`, `-p`, and `-P` options described in *BCL Converter Options* on page 40.

Convert BCL into BaseCalls Directory

The recommended workflows for converting .bcl files into *_qseqs.txt files are as follows:

1. Move to the BaseCalls directory:
   ```
   cd path/to/BaseCalls
   ```
2. Run the setupBclToQseq.py script to generate the make file:
   ```
   {path}/bin/setupBclToQseq.py -b path/to/BaseCalls
      -o path/to/output/directory
   ```
3. Run the make file:
   ```
   make -j 8
   ```

Convert BCL and Run GERALD

The `--GERALD` option allows setting up alignments. This requires GERALD.pl (from CASAVA 1.7 or above) to be accessible via the PATH environment variable.

Typical usage is:

1. Move to the BaseCalls directory:
   ```
   cd path/to/BaseCalls
   ```
2. Run the setupBclToQseq.py script to generate the make file:
   ```
   {path}/bin/setupBclToQseq.py -b path/to/BaseCalls
      -o path/to/output/directory --GERALD path/to/config.txt
   ```
3. Run the make file:
   ```
   make recursive -j 8
   ```
   Use `recursive` to recursively run make in the GERALD sub directories

BCL Converter Help

The list of command line options is available with:

```
bin/setupBclToQseq.py --help
```

BCL Converter Options

The command line options are explained below.

-b, --base-calls-directory

The base calls directory (or Bustard directory) containing the Lane subdirectories with the bcl files. This parameter is required.

-o, --output-directory

The user-selected output directory. It can be any non-existing directory, or an existing one when the `--overwrite` parameter is used. You need to either specify this parameter, or use the parameter `--in-place`, which will write output directly to the BaseCalls folder.

--overwrite

Forces the tool to accept using an existing directory as output directory. It will overwrite the previous analysis in that existing folder.

--in-place

This option will write output directly to the BaseCalls directory (`--base-calls-directory`), because it takes -b (--base-calls-directory) and turns it in the output directory. You need to either use this parameter, or specify the output directory using the parameter `-o` or `--output-directory`.

-f, --filter-directory

The directory containing the filter files. This option is only necessary if the filter files are not located in the directory OLB expects, based on the version of RTA OLB has detected. If the `-f` option is used, OLB will automatically assume the filter files are in the format generated by OLB v1.8 and RTA v1.8.

-p, --positions-directory

The directory that leads to the position files. Default value is the standard location that is produced by the version of RTA that OLB has detected.

For RTA v1.8 or older, this is the directory that contains the *_pos.txt files. For RTA v1.9 or v1.10, this is the directory that contains the lanes directories that store the *.locs or *.clocs files.

-P, --positions-format

The format of the positions files. Allowed values: -P _pos.txt, -P .locs, and -P .clocs.

Default value is the standard type of position file that is produced by the version of RTA that OLB has detected.

-i, --intensities-directory

The path to the Intensities directory; only change if the Intensities directory is not located directly above BaseCalls folder.

-c, --include-controls

By default, OLB reports controls as reads that did not pass filter, so that no downstream analysis is affected. This option can be used by OLB to report controls using the last

field in the _qseq.txt file, as described in *Converter Output Files* on page 43. This option should not be used for files that will be analyzed by any version of CASAVA.

`--ignore-missing-bcl`

If any *.bcl files are missing, OLB will not continue by default. This option tells OLB to continue even if *.bcl files are missing. Instead of a base call, OLB writes "." in the _qseq.txt file as the base for that cycle.

`--ignore-missing-stats`

If any *.stats files are missing, OLB will not continue by default. This option tells OLB to continue even if *.stats files are missing. OLB will use the value 0 for any stat that was supposed to come from the missing stats file for that cycle.

`-j, --jobsLimit`

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 -j n.

`--GERALD`

Configures the alignment sub-directory (similar to the equivalent option in OLB). This requires the PATH environment variable to contain the full path to the CASAVA bin directory. This is a required option if performing alignment.

`--help`

Use the `--help` command line option for more information.

Converter Output Files

The BCL Converter generates the following output files:

- *_qseq.txt base call and quality score files
  The main sequencing output files for the BCL Converter are the _qseq files. They contain base calls and quality scores per read, and are stored in the BaseCalls directory. The *_qseq.txt files have the following format:

Field	Description
Machine name	Identifier of the sequencer.
Run number	Number to identify the run on the sequencer.
Lane number	Positive integer (currently 1-8).
Tile number	Positive integer.
X	X coordinate of the spot. Integer.
Y	Y coordinate of the spot. Integer.
Index	Index sequence or 0. For no indexing, or for a file that has not been demultiplexed yet, this field should have a value of 0.
Read Number	1 for single reads; 1 or 2 for paired ends or multiplexed single reads; 1, 2, or 3 for multiplexed paired ends.
Sequence	Called sequence of read.
Quality	The quality string.

Filter	Did the read pass filtering? 0 - No, 1 - Yes.
	If the `--include-controls` option is used (see *BCL Converter Options* on page 40), this field reports passing filter and control status as follows:
	• The first byte is an unsigned char which indicates the identity of the control to which the read was matched, or is zero if the read wasn't identified as a control.
	• The second byte has bit flags that are used as follows:
	  • Bit 0 is pass or failed filter
	  • Bit 1: Read was flagged as a control
	  • Bit 2: Control match was ambiguous
	  • Bit 3-7: Reserved for future use

  As of RTA v1.6, OLB v1.6, and CASAVA v1.6, the X and Y coordinates for each clusters are calculated in a way that makes sure the combination will be unique. The new coordinates are the old coordinates times 10, +1000, and then rounded.
- `*_all.txt stat files.`
  The *_all.txt file contains the mean value of each channel for all cycles. The files can be found in the SignalMeans subdirectory of the BaseCalls directory.
- The BCL Converter also outputs all the plots that are in a normal Bustard directory (see the *OLB User Guide* for a description).
- The BCL Converter generates a BustardSummary.xml file.
- The BCL Converter also copies the RTA Phasing.xml file.

These output files can now be used by the demultiplexer or go straight into GERALD. They may also be used in third party software that accepts *_qseq.txt files.

Appendix A

Requirements and Software Installation for OLB

Network Infrastructure

These large data volumes mean that you will need:

1. A high-throughput ethernet connection (1 Gigabit recommended) or other data transfer mechanism.
2. A suitably large holding area for the 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 store long-term and what storage infrastructure you want to provide. OLB provides the option to perform loss-less data compression.

Storage Configurations

You can configure your analysis server with 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 OLB 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. OLB 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 sequencing platform, 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 notes to be as efficient as a standalone server. By adding computational nodes, the cluster can service more instruments.

Analysis Computer

OLB may run on any 64-bit Unix variant, if all of the prerequisites described in this section are met. However, Illumina does not support any platform other than RedHat Enterprise Linux 5.x and it's free alternative, CentOS.

Illumina recommends the IlluminaCompute data processing solution for OLB. IlluminaCompute is available as a multi-tier option, with the volume of instrument data output per week determining the recommended Tier level. For more information, contact Illumina Support.

For example, for a laboratory generating 200 GB of sequence per week, the Tier 1 IlluminaCompute solution is recommended, for which the specifications are listed below (non-IlluminaCompute systems satisfying these requirements are also fully supported):

- 1 APC Netshelter: 40U Rack with 1U KMM console
- 3 Dell R610 Server: 8 CPU cores, 48 GB RAM
- 3 Isilon IQ12000x storage modules
- 1 Serial MGT Console 16
- 2 Cisco 3750e switches

OLB 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 in OLB* on page 51.

Installation Prerequisites

The following software is required to run the Off-Line Basecaller:

- Perl 5.8 or later; it's best to install the RedHat rpms for Perl.
- Python 2.4 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
- xsltproc
- SMTP server (for optional automated email run reports)
- zlib
- bzlib

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

- gcc >= 4.0.0, except 4.0.2 (including g++)
- headers (-devel RPMs) for the required tools and libraries
- 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

NOTE
On some systems (including BSD), the ncurses headers might be required.

If you are running the Linux distribution Red Hat, the required dependencies listed above are satisfied by the Red Hat packages perl-XML-Dumper, perl-XML-Grove, perl-XML-LibXML, perl-XML-NamespaceSupport, perl-XML-Parser, perl-XML-SAX, perl-XML-Simple, perl-XML-Twig, gnuplot, ImageMagick, ghostscript, libxml2, libxml2-devel, libxml2-python, ncurses, ncurses-devel, gcc, gcc-c++, libtiff, libtiff-devel, bzip2, bzip2-devel, zlib, zlib-devel, PyXML as well as their respective prerequisites. fftw3 needs to be downloaded separately and installed from source.

Boost Libraries

OLB uses the boost libraries version 1.42.0, which is included with the distribution tarball. By default, the OLB installer will built the required components of the boost libraries, but for this it is important to make sure that the BOOST_ROOT and BOOSTROOT environment variables are unset:

```
unset BOOST_ROOT
unset BOOSTROOT
```

If boost 1.42.0 is already installed on your system, and if you want to use it instead of the version provided in the OLB tarball, simply set BOOST_ROOT to the directory where boost 1.42.0 is already installed before starting the installation of OLB.

Installing the OLB Software

To install OLB, 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 appropriate bin and lib subdirectories, which contain the scripts and makefile generators.

1 Go to the location where you want to install OLB and type the following:
```
tar xvfz OLB-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 OLB directory and type:
```
make install
```

Compiling on Other Platforms

Compiling OLB with the current makefiles works on all platforms, including many 64-bit Linux versions and Solaris. However, if your compilation does not succeed on a less commonly used platform (like 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.

Illumina does not support any platform other than Linux.

Directory Setup

Create a directory called Instruments/<instrument_name> for each sequencing platform in the same directory as the Run Folder, where <instrument_name> is the hostname of the computer that is attached to the sequencing platform.

For example, the directory for the Run Folder /data/070813_ILMN-1_0217_FC1234 would be called /data/Instruments/ILMN-1/.

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, OLB will create one for you.

Using Parallelization in OLB

Appendix B

Introduction

One of the main considerations behind the current OLB 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, OLB also provides a series of checkpoints and hooks that enables you to customize the parallelization for your computing setup. See *Customizing Parallelization* on page 53 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 OLB 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 OLB from running. Therefore, Illumina does not recommend using "lsmake" to run OLB.

NOTE
Distributed cluster computing may require significant system administration expertise.

Illumina does not support external installations.

Customizing Parallelization

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

OLB base calling can be divided further into a series of steps with different levels of scalability where synchronization "barriers" cause OLB 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, OLB 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.

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 <mycomputenode1> 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 OLB 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.

NOTE
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.

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		matrix_1_ finished.txt ...	Matrix/s_1_0001_02_mat.txt ... (more tiles/cycles)
Check File		matrix_1_ finished.txt ...	Matrix/s_1_0001_02_mat. txt ... (more tiles/cycles)

Parallelization Level	Run	Lane	Tile
Hook		cmdb5	(none)
Target	matrix		
Check File	matrix_ finished.txt		
Hook	cmdb6		
Target		phasing_1_ finished.txt ...	Phasing/s_1_0001_01_phasing. txt ... (more tiles/cycles)
Check File		phasing_1_ finished.txt ...	Phasing/s_1_0001_01_phasing. txt ... (more tiles/cycles)
Hook		cmdb1	(none)
Target	phasing		
Check File	phasing_ finished.txt		
Hook	cmdb2		
Target		s_1 ...	s_1_0001 ...
Check File		s_1_finished.txt ...	
Hook		cmdb3	(none)
Target	all		
Check File	finished.txt		
Hook	cmdb4		

Depending on the number of reads in the sequencing run, there may be multiple tile-specific targets in the matrix and phasing estimation. Matrix estimation is typically done on the second cycle of a read, phasing estimation from the first cycle onwards.

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 OLB 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

OLB requires a minimum of 2 GB RAM available per concurrent process. For most OLB tools, the amount of memory requires is linear in the number of clusters per tile and the number of sequencing cycles.

Index

Technical Assistance

For technical assistance, contact Illumina Customer Support.

Table 3 Illumina General Contact Information

Illumina Website	http://www.illumina.com
Email	techsupport@illumina.com

Table 4 Illumina Customer Support Telephone Numbers

Region	Contact Number
North America toll-free	1.800.809.ILMN (1.800.809.4566)
United Kingdom toll-free	0800.917.0041
Germany toll-free	0800.180.8994
Netherlands toll-free	0800.0223859
France toll-free	0800.911850
Other European time zones	+44.1799.534000
Other regions and locations	1.858.202.ILMN (1.858.202.4566)

MSDSs

Material safety data sheets (MSDSs) are available on the Illumina website at http://www.illumina.com/msds.

Product Documentation

If you require additional product documentation, you can obtain PDFs from the Illumina website. Go to http://www.illumina.com/support/documentation.ilmn. When you click on a link, you will be asked to log in to iCom. After you log in, you can view or save the PDF. To register for an iCom account, please visit https://icom.illumina.com/Account/Register.

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