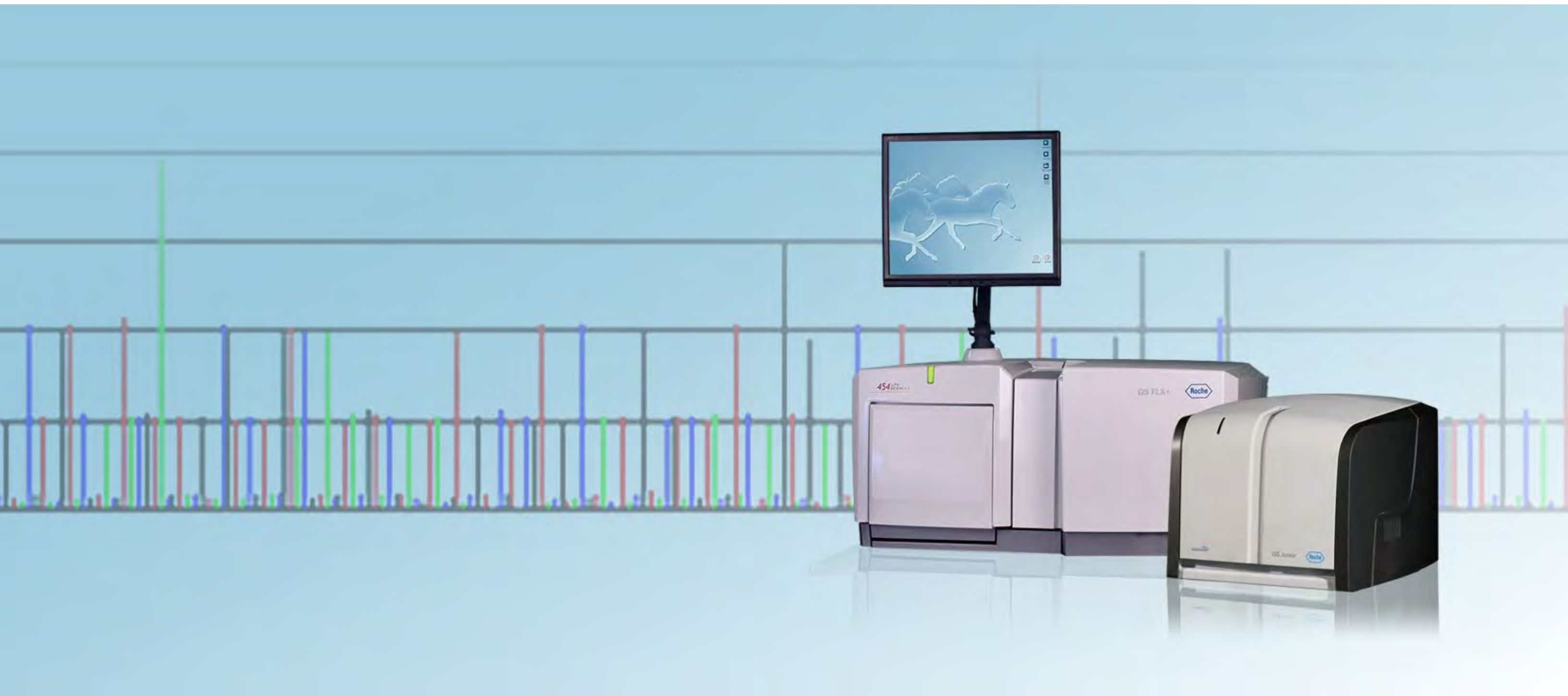
454 Sequencing System Software Manual Version 2.9

General Overview and Data File Formats

June 2013

	Instrument	Kit
✓	GS Junior	Junior
✓	GS FLX+	XL+
✓	GS FLX+	XLR70
✓	GS FLX	XLR70

For life science research only. Not for use in diagnostic procedures.



General Overview and Data File Formats

PREFACE

About this Manual

The *454 Sequencing System Software Manual* describes the software package of the GS FLX, GS FLX+ and GS Junior systems for DNA Sequencing developed by 454 Life Sciences Corporation. It is divided into 5 parts:

- General Overview and Data File Formats
- Part A: GS Sequencer and GS Junior Sequencer
- Part B: GS Run Processor, GS Reporter, GS Run Browser, and GS Support Tool
- Part C: GS *De Novo* Assembler, GS Reference Mapper, and SFF Tools
- Part D: GS Amplicon Variant Analyzer

Since the software is generally common for all systems, the manual covers all applications. The only exception to this rule is the GS Sequencer or GS Junior Sequencer application, so users should make sure to refer to the appropriate sections of Part A of the manual.

In this documentation, the phrase **GS FLX+ system** refers to the whole system for DNA sequencing developed by 454 Life Sciences Corp., including either the GS FLX or GS FLX+ Instrument, all the kits for the preparation, amplification and sequencing of a DNA (or RNA) sample, the methods to use the kits as described in the Manuals and Guides, and the software provided to process and analyze the data from sequencing runs. Likewise, **GS Junior system** refers to our similar lower throughput system based on the GS Junior Instrument. **GS FLX system** refers specifically to the GS FLX Instrument (not upgraded), and is used only when this instrument must be distinguished from the GS FLX+ Instrument. The phrase **454 Sequencing system** refers to the common technology that underlies all these systems. 454 Life Sciences Corporation is a Roche company.

Following an overview of data processing and analysis in the 454 Sequencing system, this manual provides a full description of these applications and commands, including how they are invoked through their Graphical User Interface (GUI) and at the UNIX command line level on the GS Junior attendant PC or an external computing resource (a datarig, such as the GS FLX+ computing station), and information on the format of the output files of all the applications.

System Protection

System Protection: Connection to computer networks contains an inherent risk of infection by viruses and worms and of malicious targeted attacks through the network. It is the customer's responsibility to protect the system against such threats, *i.e.* by keeping up-to-date the protection of any network to which the customer chooses to connect any instrument or computing resource. This protection might include measures such as a firewall to separate these devices from uncontrolled networks, as well as measures to ensure that the connected network is free of malicious code.

Assistance

If you have questions or experience problems with the 454 Sequencing system, please call, write, fax, or e-mail us. Be prepared to provide the serial number of your instrument and/or lot number of the kit(s) you are using. The instrument's serial number is located on the label found on the back of the instrument.

If you are located in…	Please contact Roche Applied Science Technical Support via:	
USA or Canada	**Phone:** Region .... (toll free) 1-800-262-4911	**e-mail:** us.gssupport@roche.com
Europe, Middle East, or Africa	**Phone:** Region ....+49-8856-60-6457 .... or (toll free) +800SEQUENCE	**e-mail:** service.sequencing@roche.com
Latin America	**Phone:** Argentina .... +810 810 5650 Brazil .... 0800 772 0295 Central America & Caribbean ....800 7624 .... or 507 378 1245 Costa Rica....................................8000 762 431 El Salvador ....503 21 13 39 35 Guatemala..............................1 801 13 76 243 Honduras.................................800 22 25 06 26 ....or 504 22 25 0626 Nicaragua ....1 800 22 02 125 Panama ....800 7624 .... or 507 378 1245 Chile..+800 373 700 Columbia ....01800 011 7911 Ecuador.................................+593 2 399 7291 Mexico .... +01800 718 8853/54 Peru... +511 618 8990 Uruguay ....+598 2 619 2273 Venezuela.......................... +58 212 273 46 10	**e-mail:** ras.support@roche.com
Asia Pacific	**Phone:** Region .... (toll free) 800 820 0577 China Mainland...................... 008 018 63123 China Taiwan .... 800 966 851 China Hong Kong......................800 852 3686 Singapore.....................................1800 814 958 Malaysia ....180 064 5619 Australia ....007 988 620647 Korea .... 001 800 861 0660 Thailand ....1208 6101 Vietnam....................................1800 186 10007 Philippines .... +632 718 7506	**e-mail:** asc.support@roche.com
Japan	**Phone:** Region .... +03-5443-5287	**e-mail:** tokyo.biochemicals@roche.com

1 OVERVIEW OF THE 454 SEQUENCING SYSTEM SOFTWARE

The 454 Sequencing system, developed by 454 Life Sciences Corporation, a Roche company, is an ultra-high-throughput automated DNA sequencing system capable of carrying out and monitoring sequencing reactions in a massively parallel fashion (tens to hundreds of thousands of simultaneous reactions, in the "wells" of a PicoTiterPlate device). During DNA-directed DNA synthesis, pyrophosphate (PPi) is released with each nucleotide addition; the system's chemistry generates an amount of light commensurate with the amount of PPi released; this light is captured by a charge-coupled device camera and converted into a digital signal. (For more information on the basics of the 454 Sequencing system, please refer to the *GS Junior Instrument Owner's Manual* or the *GS FLX+ Instrument Owner's Manual*).

The software package described in this manual also includes a variety of applications that are used primarily or exclusively off-instrument (on a datarig or GS Junior attendant PC). The GS Reporter and the GS Run Browser applications are used to view and troubleshoot the results of a completed sequencing run; the GS Support Tool is used to package sequencing run data to send to Roche Customer Support for further help and troubleshooting; and the SFF Tools are a set of commands used to create, manipulate and access sequencing trace data from SFF files. However, these applications and commands are not required steps of data processing and analysis.

Data handling in the 454 Sequencing system occurs in three main phases, each governed by one or more specific applications:

- Data Acquisition
- Data Processing
- Data Analysis

1.1 Data Acquisition

The data acquisition phase is controlled by the GS Sequencer software application on the GS FLX+ Instrument or by GS Junior Sequencer on the GS Junior Instrument (this is the only application that has two different implementations under the two systems). The raw data consists of a series of digital images captured by the camera. The images are a representation of the surface of the PicoTiterPlate device over which the sequencing reactions are taking place; and each image corresponds to one nucleotide flow over that surface. If the sample DNA fragment present in a given well of the PicoTiterPlate device is extended during a nucleotide flow, light is emitted from the well and captured on the image corresponding to that flow. Furthermore, the amount of light emitted is proportional to the number of nucleotides extended.

Data acquisition occurs in a single step:

1. The GS Junior Sequencer or GS Sequencer application records a set of raw digital images representing the light detected over the surface of the PicoTiterPlate device, during each reagent flow of the sequencing run (data acquisition).

Table 1 lists the inputs and outputs of data acquisition.

Application	**Input**	**Output**	**Main processing steps**
GS Junior Sequencer or GS Sequencer	Light	Raw images	● Image acquisition and storage to disk

Table 1: The data acquisition step in the 454 Sequencing system, with its input, output, and main processing step. For a full description of the GS Junior Sequencer or GS Sequencer application, see Part A of this manual; and for the GS Run Processor application, see Part B of this manual.

1.2 Data Processing

The data processing steps are as follows:

1. The first step of the GS Run Processor application, image processing, performs initial pixel-level calculations, and then groups pixels from the image set into a representation of the PicoTiterPlate wells where sequencing reactions were detected.
2. The second step of the GS Run Processor application, signal processing, performs well-level calculations across the whole series of images to generate well “flowgrams” (and the basecalls of the DNA fragments being sequenced in all the active wells of the PicoTiterPlate device; “reads”).

Table 2 lists the inputs and outputs of data processing, as well as the individual processing steps.

Application	Input	Output	Main processing steps
GS Run Processor (image processing step)	Raw images	Composite Wells Format (CWF) Files	● Subtract background and normalize the images (at the pixel level) ● Find the active wells on the PicoTiterPlate device ● Extract the raw signals for each flow, in each active well ● Write the resulting flow signals into “composite wells format” (CWF) files
GS Run Processor (signal processing step)	Composite Wells Format (CWF) Files	Corrected CWF files, and SFF files containing read basecalls and per-base quality scores	● Filter out lower signal ghost wells ● Correct for crosstalk between neighboring wells ● Correct for known “out-of phase” errors (incomplete extension and carry-forward) ● Correct for signal droop and perform residual background subtraction ● Filter (pass or fail) the processed reads based on signal quality ● Trim read ends for low quality and adaptor sequences ● Update the CWF files with the fully processed data ● Generate Standard Flowgram Format (SFF) files containing the basecalled read sequences and per-base quality scores

Table 2: The two main components of data processing in the 454 Sequencing system, with their inputs, outputs, and main processing steps. They are performed in succession, in the order indicated; the SFF files output by the signal processing step of the GS Run Processor application are used as input to the data analysis applications (see). For a full description of the GS Run Processor application, see Part B of this manual.

Image and signal processing can be carried out either during or after a sequencing run. Figure 1 summarizes the various paths that can be followed, based on the initial choice made during run setup.

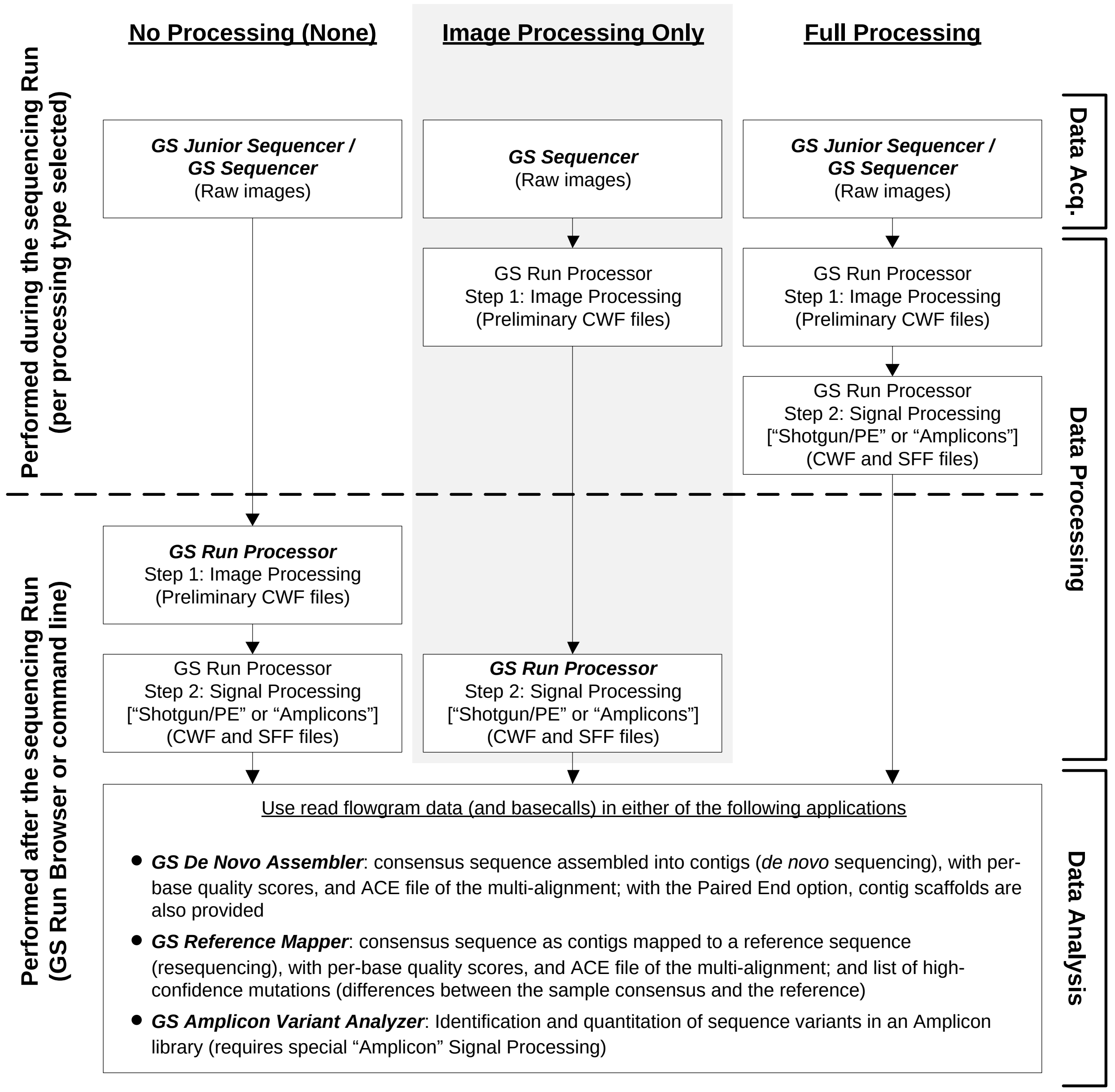

No Processing (None)
Image Processing Only
Full Processing
Performed during the sequencing Run (per processing type selected)
Performed after the sequencing Run (GS Run Browser or command line)
Data Acq.
Data Processing
Data Analysis
GS Junior Sequencer / GS Sequencer (Raw images)
GS Sequencer (Raw images)
GS Run Processor Step 1: Image Processing (Preliminary CWF files)
GS Run Processor Step 2: Signal Processing ["Shotgun/PE" or "Amplicons"] (CWF and SFF files)
Use read flowgram data (and basecalls) in either of the following applications
GS De Novo Assembler: consensus sequence assembled into contigs (de novo sequencing), with per-base quality scores, and ACE file of the multi-alignment; with the Paired End option, contig scaffolds are also provided
GS Reference Mapper: consensus sequence as contigs mapped to a reference sequence (resequencing), with per-base quality scores, and ACE file of the multi-alignment; and list of high-confidence mutations (differences between the sample consensus and the reference)
GS Amplicon Variant Analyzer: Identification and quantitation of sequence variants in an Amplicon library (requires special "Amplicon" Signal Processing)

Figure 1: Data processing paths in the GS Junior and GS FLX+ systems. The blocks identify data acquisition, data processing or data analysis applications and their outputs. Raw images are captured as part of the sequencing run (data acquisition). Depending on the processing type selected during run set up (top of each column), the image processing and signal processing steps can either be performed as part of the sequencing run (above the dotted line), or after the run (below the dotted line). Note that Image Processing Only (the shaded path) cannot be initiated in GS Junior Sequencer. Applications launched by direct user input are shown in *Bold-Italics*; others are run automatically after the previous stage is completed.

The image processing step is common to all types of sequencing runs, but signal processing is tailored to the specific application. Full processing is a combination of image processing followed by application-specific signal processing.

Users can control data processing with respect to;

- When to process (during or post sequencing run)
- Where to process (on-instrument, on a GS Junior attendant PC, or on an external datarig)
- Which steps to process (image processing, signal processing, full processing)
- What to process (standard shotgun/paired end library or amplicon library).

1.2.1 Data Processing Options During a Sequencing Run

GS Sequencer and GS Junior Sequencer each provides a choice of processing type during a sequencing run (see Part A of this manual for information on selecting processing type during run setup). The available choices will depend on available computing resources, as well as the selected sequencing kit and sequencing conditions (below). See Part B of this manual for additional technical details on the underlying data processing pipelines.

- **None** - disable data processing during the sequencing run (image acquisition only). If this is selected, all other steps of image and signal processing are carried out separately on an external datarig or GS Junior attendant PC. This maximizes the amount of time the instrument is available for sequencing. Although a backup script can be triggered, no post-analysis script is run.
- **Image processing only (not available on GS Junior Sequencer)** - process images on the instrument concurrently with image acquisition, but handle the time-consuming signal processing step on a datarig. Image processing completes by the time the instrument is ready for another run with the GS Junior and XLR70 sequencing kits, but may extend a few hours beyond the fluidics stage with the XL+ sequencing kit. A post-analysis script can be triggered, which allows pipeline-specific processing steps to be automated.
- **Full processing for Shotgun or Paired End (not available on-instrument with the XL+ sequencing kit)** - include both the image and signal processing steps during the sequencing run. This is the simplest option from the standpoint of instrument operation, as all data processing up to the generation of read flowgrams and basecalling of the reads is carried out during the run, without user intervention. However, full processing can be computationally intensive.
  - GS Junior sequencing kit: the GS attendant PC can take up to 10 hours to process a run.
  - XLR70 sequencing kit: the on-instrument computing resources can take up to 80 hours to process a 200 cycle run.
  - XL+ sequencing kit: on-instrument full processing is not supported.
- **Full processing for Amplicons (processing with the XL+ sequencing kit requires flow pattern A, and is not available on-instrument)** - include both image and signal processing steps specially tuned for amplicon data sequenced with the cyclic flow pattern. This pipeline is not tuned to long amplicon data.
- **Full processing for Long Amplicons (available only with the XL+ sequencing kit using flow pattern B, and not available on-instrument)** - include both image and signal processing steps specially tuned for long amplicon data sequenced with the acyclic flow pattern B. Three long amplicon pipeline options are available.
  - Long Amplicons #1 - similar to the standard Amplicons pipeline (full-length, bidirectional sequencing), but with higher stringency.
  - Long Amplicons #2 - similar to the Long Amplicons #1 pipeline, but more suitable for particularly long amplicons or samples with content that is difficult to sequence, as trimming is allowed.
  - Long Amplicons #3 - suitable for samples where bidirectional coverage is not required and reads can be trimmed with a relatively low stringency.

The term 'long amplicon' refers specifically to amplicons longer than ~550 bp. The three long amplicon pipelines are optimized for amplicons with a length of about 750 bp. It may be possible to obtain acceptable results with amplicons that are longer than this supported size by customizing pipeline parameters (see the section on Pipeline Stringency in Part B of this manual).

The three long amplicon pipelines are designed for use with the XL+ kit using flow pattern B. Standard amplicon processing with the XL+ kit using flow pattern A is available for testing purposes within GS Run Browser, but is not officially supported.

1.2.2 Data Processing After a Sequencing Run

If anything other than Full processing was selected during run setup, processing must be completed *via* the GS Run Processor after the run has finished, either as an option within the GS Run Browser GUI or at the 'command line' level. Data can also be reprocessed for troubleshooting purposes or to reanalyze the data set using different settings. Data processing will produce comparable results whether it is carried out during a run, or afterwards.

See Part B of this manual for additional information on GS Run Processor and GS Run Browser, and for details about the individual steps involved with image and signal processing.

The availability and support status of various pipeline choices are dependent on a wide variety of variables, including sequencing hardware, sequencing conditions, computing resources, and timing (during or after a run). Table 3 summarizes this situation.

Sequencing Kit [Cycles or Flows]	Image Processing Only	Shotgun or Paired End	Amplicons (Standard)	Long Amplicons #1, #2, & #3
GS Junior Kit [42, 100, or 200 cycles]	b	a	a	f
XLR70 Kit [100, 150, or 200 cycles]	a	a	a	e
XL+ Kit, Flow Pattern A [400 cycles]	a	c	d	e
XL+ Kit, Flow Pattern B [1779 flows]	a	c	e	c

Table 3: Availability of data processing pipeline choices. GS Sequencer and GS Junior Sequencer (described in Part A of this manual) are used to choose processing type <u>during</u> run setup. GS Run Browser (described in Part B of this manual) is used to select processing type <u>after</u> a run. Processing on-instrument refers to processing on a GS FLX+ internal server, which does not have sufficient computing power to effectively fully process runs with greater than 200 cycles (800 flows).

a	**Available during or after a sequencing run.**
b	**Not visible during, available after a sequencing run.**
c	**Not available during, available after a sequencing run [not supported on-instrument for a GS FLX+ Instrument].**
d	**Not available during, available after a sequencing run [not supported on any computing platform].**
e	**Not available during or after a sequencing run.**
f	**Not visible during, not available after a sequencing run.**

1.3 Data Analysis

The data analysis phase offers a choice of several downstream analysis paths to generate the desired final output: a consensus sequence of the DNA sample generated by the assembly of reads into contigs and scaffolds (GS *De Novo* Assembler); a consensus sequence along with a list of high-confidence differences obtained by mapping the reads to a known reference sequence (GS Reference Mapper); or the identification and quantitation of sequence variants by the ultra-deep sequencing of amplicons (GS Amplicon Variant Analyzer). All data analysis outputs also include base-per-base quality scores (Phred-equivalent) and other specific metric files.

The data analysis steps are as follows:

1. The GS *De Novo* Assembler application generates a consensus sequence of the whole DNA sample, by assembling the reads into contigs (*de novo* shotgun assembly). An option allows the use of one or more sequencing runs performed on a paired end library (any type, or even a combination of paired end library types) prepared from the same DNA sample, to be analyzed together with shotgun sequencing run(s) and help order and orient the resulting contigs into scaffolds. (paired end reads do not necessarily need to be analyzed together with shotgun reads.)
2. The GS Reference Mapper application generates the consensus DNA sequence by mapping, or aligning, the reads to a reference sequence; as well as a list of high-confidence SNPs or differences (individual bases or short blocks of bases (less than ~70 bases) that differ between the consensus DNA sequence of the sample and the reference sequence). Robust cDNA analysis is also available.
3. The GS Amplicon Variant Analyzer application compares reads from an amplicon library to corresponding reference sequences, and allows the user to detect, identify and quantitate the prevalence of sequence variants.

The data analysis applications use the fully processed and 'trimmed' read basecalls of a sequencing run, or of a pool of runs, to produce initial alignments to the reference sequence (or read-to-read overlaps for the GS *De Novo* Assembler). They then use a combination of nucleotide and flowgram information for consensus-calling of the contigs and determination of quality values for the contig sequences. Contig consensus-calling is carried out in 'flowspace' (*i.e.* it operates directly on the processed signals measured from the wells), followed by basecalling to produce a consensus sequence for the sample. Table 4 lists the specific outputs of the three data analysis applications as well as the individual functions carried out by each one.

The final output of the 454 Sequencing system thus varies depending on what kind of analysis is performed: Assembly, Mapping or Amplicon Variant Analysis (or no analysis of any kind). In all cases, however, the output DNA sequence is supplied as a set of FASTA files, with associated "Quality Scores" and other run and data metrics files useful for troubleshooting and determining the overall quality of the sequencing run. ACE-formatted files are also produced by each of the data analysis applications to allow users to view alignment results using third-party software tools.

Table 4 lists the inputs and outputs of data processing, as well as the individual processing steps.

Application	Input	Output	Main processing steps
GS *De Novo* Assembler	SFF files, from one or multiple sequencing runs, containing read flowgrams and basecalls, and per-base quality scores	Sample consensus sequence, assembled *de novo* (and scaffold information, with paired end option)	Identify pairwise overlaps between reads, in nucleotide space Construct multiple alignments of reads that tile together (*i.e.* form contigs), based on the pairwise overlaps Generate consensus basecalls of the contigs by averaging the processed flow signals for each nucleotide flow included in the alignment, in flowspace Output the contig consensus sequences and corresponding quality scores, along with an ACE file of the multiple alignments and assembly metrics files Additional steps with paired end option: Identify pairwise overlaps between paired end tags and the shotgun contigs Organize the contigs into scaffolds (order, orientation, and approximate distance) Output the scaffolded consensus sequences and corresponding quality scores, along with an AGP file of the scaffolds and specific metrics Tables
GS Reference Mapper		Sample consensus sequence, mapped to a reference sequence; and list of differences	For each read, search for alignment(s) to the reference sequence, in nucleotide space Construct contigs and compute a consensus basecall sequence from the signals of the aligned reads (flowspace) Identify the positions where the consensus or subsets of the reads that comprise it differ from the reference sequence (or reads from one another); these are the "putative differences" Evaluate the putative differences to identify high-confidence differences Output contig consensus sequence(s) and corresponding quality scores, an ACE file of the multiple alignments of the reads and contigs to the reference, the list of identified differences, and mapping metrics files
GS Amplicon Variant Analyzer		Identity and quantitation of sequence variants	Trim reads (remove adaptor sequences) Assign reads to "Samples" (demultiplex data sets) Align Sample reads to their reference sequences Quantitate variant frequency for each Sample

Table 4: The three applications of the data analysis phase of the 454 Sequencing system, with their inputs, outputs, and main processing steps. Note that all data analysis applications use as input the reads and flowgrams output in SFF format by the data processing (GS Run Processor application). For a full description of the various data analysis applications, see Parts C and D in this manual.

The GS *De Novo* Assembler, GS Reference Mapper and GS Amplicon Variant Analyzer applications, and the SFF Tools commands, are *always* run separately from the sequencing run (*i.e.* off-instrument, on a datarig or GS Junior attendant PC). The rationale for this is that these applications either are not usually applied to individual runs but rather draw on multiple runs, or require additional information beyond the run data (such as a reference genome against which to map the sequencing reads). In addition, these applications can take hours to complete, during which time the GS FLX+ or GS Junior Instrument would not be available for another run. Run troubleshooting may also be time intensive, so the GS Reporter and the GS Run Browser applications are provided both on the GS FLX+ Instrument or GS Junior attendant PC, and as separate applications that can be run on a datarig.

1.4 Data Output and Folder Structure

This section provides an overview of the listing and organization of the files that are generated at each step of data processing or data analysis, and made available to the user.

After a sequencing run, results can be found in the "*/data*" directory on the GS FLX+ Instrument or the GS Junior attendant PC, grouped by date, where each "date" folder contains individual run folders. Each run folder is identified by the run name specified during run set up (*i.e.* with an 'R_' prefix, denoting 'R'un; see first "Note" below). Run folders contain the raw data of the sequencing run (*i.e.* the results of the GS Sequencer or GS Junior Sequencer application), as well as any Data Processing folder(s) for that run (identified by a 'D_' prefix, for 'D'ata). Data Processing folders, in turn, contain the results of the image processing and/or signal processing steps.

<u>"tmp" sub-directory required</u>: The */data* directory in the GS FLX+ Instrument on-instrument computer or the GS Junior attendant PC contains a tmp sub-directory. Do not delete the tmp sub-directory as it is required for the proper functioning of the instrument software.

Because the data analysis applications are typically performed on a pool of sequencing runs rather than on any single run, their results are not associated with a specific sequencing run or data processing invocation; rather, assembly, mapping and amplicon variant analysis results are deposited in the "current working directory" at the time the application is launched, or written to a directory specified by the user (see section 1.4.3). The SFF Tools commands also usually deposit their output into the "current working directory" or a directory specified on the command line.

- The user enters a unique name for the run during set up. A good choice for a unique name could be structured as follows: PicoTiterPlate device size and barcode #, kit lot #, genome, run #; *e.g.* "70x75_123456_081708_ecoli_Run1". To form the complete name of the run files, the date stamp, time stamp, instrument name, and user name will be added automatically in front of this unique run name. The name structure is:

  "R_yyyy_mm_dd_hh_min_sec_machineName_userName_uniqueRunName".

- Before running any data processing or data analysis application (GS Run Processor, GS *De Novo* Assembler, GS Reference Mapper, or GS Amplicon Variant Analyzer), the necessary input data for the sequencing run(s) being processed or analyzed (see Table 1 and) must be made available to the datarig or GS Junior attendant PC where the application is located. The GS FLX+ Instrument or GS Junior attendant PC can be configured to automatically transfer the output files that result from the GS Run Processor application to a remote disk or datarig (see the section on data transfer scripts in the *GS FLX+ System Site Preparation Guide* or the *GS Junior System SysAdmin Guide*). Alternatively, the Data tab of the GS Sequencer application on the GS FLX+ Instrument, or of the GS Junior Sequencer application on the GS Junior attendant PC, can be used to transfer the raw images of sequencing run(s) to a pre-configured destination. For more information on the Data tab, see Part A of this manual.

1.4.1 Data Acquisition (GS Sequencer and GS Junior Sequencer) Results: the Run Folder

The organization of a generic run folder ('R_') is depicted in Figure 2. All the raw data (raw images, log files, *etc.*) remain in temporary local storage on the GS FLX+ Instrument or the GS Junior attendant PC in case the user chooses to re-analyze them (*e.g.* using the reanalysis function of the GS Run Browser; see Part B, Section 3 of this manual). In addition, if "Backup" is selected during run set up, raw and processed data files from the run can be transferred to a network location specified by the *System Administrator*, for long-term storage.

```
R_yyyy_mm_dd_hh_min_sec_machineName_userName_uniqueRunName/
   dataRunParams.parse
   imageLog.parse
   PTP_flowOrder_cycleCount.icl
   runlog.parse
   rawImages/
      00001.png
      00002.png
      00003.png
      …
```

Figure 2: General organization of a "run" folder. On the GS FLX+ Instrument or the GS Junior attendant PC, this is located inside /data/*date*; while on a datarig it can be placed anywhere. Words in *italics* are generic.

Several other files or directories may appear in the R_ directory, which are created for internal use of the software. They are not described in this manual and may indeed be reorganized or eliminated in further releases of the software; they may include the following:

- Additional files:
  - aaLog
  - fpgaReadWriteLog.bin
  - flowCalibrtionLog.nfc
  - tempControlLog.ntc
  - dmesg.txt
  - debugMessageLog.txt
- Additional directories:
  - calibrate
  - prime
  - prewash

1.4.2 Data Processing (GS Run Processor) Results: the Data Processing Folder

The organization of a generic Data Processing folder ('D_') is depicted in Figure 3. D_ folders are created by the GS Run Processor application, within the R_ folder of the sequencing run whose data is being processed. Since a data set can be re-processed multiple times (*via* the GS Run Browser; see Part B, Section 3 of this manual), a given R_ folder can contain multiple D_ folders. To the extent that they are generated during a sequencing run, per the processing type selected (see section 1.2), all the processed data (basecalls and quality scores, run metrics, log files, *etc.*) remain in temporary local storage on the GS FLX+ Instrument or the GS Junior attendant PC. In addition, if "Backup" is selected during run set up, raw and processed data files from the run can be transferred to a network location specified by the *System Administrator*, for long-term storage. See Part B, Section 1 for full file descriptions and for more information on the GS Run Processor application.

```
R_yyyy_mm_dd_hh_min_sec_machineName_userName_uniqueRunName/1
   […]
   D_yyyy_mm_dd_hh_min_sec_machineName_analysisType/2
      gsRunProcessor.log
      gsRunProcessor_err.log
      dataRunParams.xml
      regions/2
         region.cwf2
      sff/2
         uaccnoRegion.sff2
```

Figure 3: General organization of a "Data Processing" folder. Data Processing ('D_') folders are created within the corresponding run's 'R_' folder; an R_ folder can contain multiple D_ folders, if the data set was re-processed. Words in *italics* are generic. The superscripts indicate the application by which the folders and files are generated: GS Sequencer or GS Junior Sequencer^1, GS Run Processor^2. The set of SFF files are generated during the signal processing step of the GS Run Processor, using the "universal" accession prefix described in section 2.3.7. See Part B, Section 1 of this manual for full file descriptions.

The GS FLX+ Instrument or the GS Junior attendant PC processes the sequencing data "on-the-fly," *i.e.* the data is processed (to the extent specified in the processing type selected) and deposited in the Data Processing folder, concurrently with the run. When the "Run Completed" window appears on the screen, the processing of the sequencing run has completed and the results are ready for further processing or transfer.

1.4.3 Data Analysis Applications Results

As indicated above, the data analysis applications are often performed on a pool of sequencing runs rather than on any single run, and/or can require additional information beyond the run data. For this reason, they are carried out off-instrument, *via* a Graphical User Interface (GUI) or from the command line on a datarig, rather than on the GS FLX+ Instrument. For the GS Junior system, this can also be performed on a separate computer resource (datarig) but it is typically performed on the attendant PC, though still separately from the data acquisition / data processing.

As a consequence of this separation, the result files generated by these applications are not deposited in a run folder. For the GS *De Novo* Assembler and the GS Reference Mapper, rather, one of the following will apply:

- A folder with a 'P_' prefix (for 'P'ost-Run Analysis) is created to receive them, in the user's current working directory on the datarig or GS Junior attendant PC at the time the application is launched, or written to a directory specified by the user *via* the applications' GUI or on the command line.
- Mapping and Assembly can also be carried out in a "project-based" fashion, whereby data sets can be added to existing results (or a new reference sequence can be specified) for an existing Assembly or Mapping "project". This uses the corresponding applications' GUI (or can be done using the newAssembly, newMapping, and associated commands), and the data is then stored in a "Project" folder. A Project folder is identified by the "454Project.xml" file it contains.

The folder and file structures generated for each of these commands (or GUI equivalents) are shown in the Figures below: the "one-step" runAssembly (Figure 4) or runMapping (Figure 5) commands, for the "standard", non-incremental assembly or mapping of one or more runs; and newAssembly (Figure 6) or newMapping (Figure 7), for the "project-based", incremental assembly or mapping of one or more runs. See Part C, Sections 1 and 2 of this manual for full file descriptions and for more information on the GS *De Novo* Assembler and GS Reference Mapper applications.

The GS Amplicon Variant Analyzer application can also operate either *via* a GUI or *via* its own Command Line Interface (the AVA-CLI), but its output is not structured like that of the other two data analysis applications. See Part D of this manual for details on this application.

```
$[current_working_directory] or [directory_specified]
   P_yyyy_mm_dd_hh_min_sec_runAssembly/
      454AllContigs.fna
      454AllContigs.qual
      454LargeContigs.fna
      454LargeContigs.qual
      454NewblerMetrics.txt
      454NewblerProgress.txt
      454AlignmentInfo.tsv
      454ContigGraph.txt
      454PairAlign.txt
      454ReadStatus.txt
      454Contigs.ace or ace/ContigName.ace or consed/…
      sff/
         *.sff
      454Scaffolds.fna1
      454Scaffolds.qual1
      454Scaffolds.txt1
```

Figure 4: File output of the GS *De Novo* Assembler application, using the runAssembly command (or its GUI equivalent). All result files (specifying the actual *contig names*) are placed in a folder with a 'P_' prefix, within the user's current working directory when running the command or in a directory specified by the user. All input SFF files used in the assembly are organized in the sff sub-directory. [1]These files are produced only when the paired end option is used; the paired end option also adds sections to the 454NewblerMetrics.txt file. See Part C, Section 1 of this manual for full file descriptions.

```
$[current_working_directory] or [directory_specified]
   P_yyyy_mm_dd_hh_min_sec_ runMapping /
      454AllContigs.fna
      454AllContigs.qual
      454LargeContigs.fna
      454LargeContigs.qual
      454AllDiffs.txt
      454HCDiffs.txt
      454NewblerMetrics.txt
      454NewblerProgress.txt
      454MappingQC.xls
      454AlignmentInfo.tsv
      454PairAlign.txt
      454ReadStatus.txt
      454RefStatus.txt
      454Contigs.ace or ace/refaccno.ace or consed/…
      sff/
         *.sff
```

Figure 5: File output of the GS Reference Mapper application, when using the runMapping command (or its GUI equivalent). All result files (specifying the actual *reference sequence accession numbers*) are placed in a folder with a 'P_' prefix, within the user's current working directory when running the command or in a directory specified by the user. All input SFF files used in the mapping are organized in the sff sub-directory. See Part C, Section 2 of this manual for full file descriptions.

```
$[current_working_directory] or [directory_specified]
   P_yyyy_mm_dd_hh_min_sec_runAssembly/
      assembly/
         454AllContigs.fna
         454AllContigs.qual
         454LargeContigs.fna
         454LargeContigs.qual
         454NewblerMetrics.txt
         454NewblerProgress.txt
         454AlignmentInfo.tsv
         454ContigGraph.txt
         454PairAlign.txt
         454ReadStatus.txt
         454Contigs.ace or ace/ContigName.ace or consed/…
         454AssemblyProject.xml
         454Scaffolds.fna1
         454Scaffolds.qual1
         454Scaffolds.txt1
      sff/
         *.sff
      454Project.xml
```

Figure 6: File output of the GS *De Novo* Assembler application, when using the newAssembly and related commands (or their GUI equivalents) for "project-based" assembly. All result files (specifying the actual *contig names*) are placed in a folder within the user's current working directory when running the command or in a directory specified by the user. This is identified as a "Project" folder by the presence of a 454Project.xml file within it. All assembly status and result files are organized in the assembly sub-folder; and all input SFF files used in the assembly (or symbolic links to them) are organized in the sff sub-folder. [1]These files are produced only when the paired end option is used; the paired end option also adds sections to the 454NewblerMetrics.txt file. See Part C, Section 1 of this manual for full file descriptions.

```
$[current_working_directory] or [directory_specified]
   P_yyyy_mm_dd_hh_min_sec_ runMapping/
      mapping/
         454AllContigs.fna
         454AllContigs.qual
         454LargeContigs.fna
         454LargeContigs.qual
         454AllDiffs.txt
         454HCDiffs.txt
         454NewblerMetrics.txt
         454NewblerProgress.txt
         454MappingQC.xls
         454AlignmentInfo.tsv
         454PairAlign.txt
         454ReadStatus.txt
         454RefStatus.txt
         454Contigs.ace or ace/ContigName.ace or consed/…
         454MappingProject.xml
      sff/
         *.sff
      454Project.xml
```

Figure 7: File output of the GS Reference Mapper application, when using the newMapping and related commands (or their GUI equivalents) for "project-based" mapping. All result files (specifying the actual *reference sequence accession numbers*) are placed in a folder within the user's current working directory when running the command or in a directory specified by the user. This is identified as a "Project" folder by the presence of a 454Project.xml file within it. All mapping status and result files are organized in the mapping sub-folder; and all input SFF files used in the mapping (or symbolic links to them) are organized in the sff sub-folder. See Part C, Section 2 of this manual for full file descriptions.

2 DATA FILES AND FORMATS

Section 1.4 lists all the files and folders that constitute the deliverable output of the 454 Sequencing system data processing and data analysis software for a generic sequencing run, including the results of the GS *De Novo* Assembler and GS Reference Mapper applications. The actual directories generated may contain a number of additional files, but those are intermediate or log files generated for use only by Roche Customer Support personnel, in the event that a run might require additional investigation.

The 454 Sequencing system software uses fixed names for the files it generates, and the structure and names of the directories allows to differentiate individual sequencing runs or post-run analyses. This section describes the nomenclature conventions and file formats used by the software. Examples of the various file types are given in the Sections of this manual that describe each application in detail.

Note that the content of this Section does not apply to the GS Amplicon Variant Analyzer software, whose output structure is completely distinct (beyond the basic data processing files and folders).

2.1 Directory Naming Conventions

When a sequencing run is performed on a GS FLX+ Instrument or a GS Junior Instrument, its results are placed in a run folder, where the format of the run name is generated by the software and includes the following components:

R_year_month_day_hour_minute_second_instrument_user_runname

A similar naming convention is used for the Data Analysis folder(s) which are deposited inside the corresponding run folder by the run-time data processing applications (either on the GS FLX+ Instrument, a GS Junior attendant PC, or a datarig). Data Analysis folders contain all the flow signal and signal processing files. Their names include the following components (except that “instrument” and “user” are not included when using the command line software):

D_year_month_day_hour_minute_second_instrument_user_analysisname

Finally, the GS *De Novo* Assembler and GS Reference Mapper applications create a “Post-Run Analysis” sub-directory to include all the files they generate, using the following naming convention:

P_year_month_day_hour_minute_second_runCommandName

The rest of the files within these directories (or within the current working directory) have either fixed names or simple, standard naming conventions (*e.g.* files specific to a region or key are named with the region or key at the beginning of the name). Exact nomenclature for all the files and other sub-folders produced by the data processing applications are provided in the “output” subsection of the description of each application, in the various Sections of Parts A, B, C, and D of this manual.

2.2 Format Requirements for Input FASTA and FASTQ Files

The GS *De Novo* Assembler and the GS Reference Mapper can take FASTA and FASTQ files as input, including FASTQ files obtained through Short Read Archive (SRA) or other sources.

2.2.1 FASTA File Format

For the reference or input read FASTA file(s) to be readable by the 454 Sequencing system software, they must follow the industry standards for a FASTA file. In particular:

- The first line (descriptor line) of each sequence entry in the file should begin with a '>'.
- There may be one or more additional header lines for a sequence entry, each beginning with a '>' or ';' character. The first line not beginning with a '>' or ';' starts the sequence region of the entry.
- The sequence region may contain any characters, but only the alphabetic characters will be used to form the sequence. All alphabetic characters are converted to uppercase, and any alphabetic character that is not A, C, G or T will be treated as an N.
- Multiple sequences may be included in the file (each starting with a '>').
- Only the characters between the '>' and the first whitespace character are used to identify the sequence (*i.e.*, the "accno" for the sequence). For clarity, each sequence in a project should be identified uniquely within the characters prior to the first whitespace in their respective descriptor lines.

For example:

```
>Ecolik12     4300K bp
CCTTGTGCAGTAGCACTTAATCATCATGTTTTAGCATTTTGATCTTCTGCTCAATTTCTT
AAGCTAGACGCTCAATCTTCTTATGATGAACGATTTCTTCTTCATGGTGTTTTTTCATAT
......
```

2.2.2 FASTQ File Format

A FASTQ file includes quality scores associated with each position in a sequence. For input read FASTQ file(s) to be readable by the 454 Sequencing system software, they must follow the industry standards for a FASTQ file. In particular:

- The first line (descriptor line) of each sequence entry in the file should begin with a '@', followed by a sequence identifier and an optional description.
- Line 2 is the raw sequence, formatted in the same way as a FASTA file, but without line breaks.
- Multi-line sequences are not supported.
- Line 3 begins with a "+", optionally followed by the sequence identifier and any description.
- Line 4 encodes the quality values for the sequence in Line 2, and must contain the same number of symbols as letters in the sequence.

For example:

```
@SEQ_ID Ecolik12     4300K bp
CCTTGTGCAGTAGCACTTAATCATCATGTTTTAGCATTTTGATCTTCTGCTCAATTTCTT
+
!''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65
```

Reads retrieved from an SRA archive are in FASTQ file format, with specific naming conventions that allow automatic detection of paired end libraries. Do not rename FASTQ files from an SRA archive. Use the `-lib` option with the addRun command to manually specify paired end libraries for non-SRA archive FASTQ files.

The expected orientations of the halves of a Paired-end read are described in Section 4.8 in Part C of this manual. If the orientations of the halves of paired reads in the FASTQ files deviate from the expected orientations, the reads must be converted to FASTA format with the appropriate paired-end annotations (see Section 4.13.3 in Part C of this manual). In addition, it may be necessary to reverse-complement one or both halves.

2.3 Standard File Formats

Most of the file formats are specific to the type of data being stored, such as the image files or the wells data files. Other files adhere to standard formats used throughout the generation and processing of sequencing run data, assembly data and mapping data. Example files in many of these formats are provided in the Output sub-sections of the applications' descriptions.

2.3.1 Composite wells file format

The CWF file is a container format which stores multiple "streams" of information. The container itself is a "ZIP" file (http://www.pkware.com/documents/casestudies/APPNOTE.TXT) with a single level hierarchy. Each stream is named and compressed separately, allowing for rapid access to any information in the file. The CWF file format is inspired from the "OpenDocument format" described in ISO/IEC 26300:2006 (http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43485). The OpenDocument format benefits from the segregation of concerns by separating the content, styles, metadata and application settings into four separate XML streams; the CWF format maintains a similar separation: flowgrams, called bases, meta data, processing history and run metrics are stored individually. The user should not normally unpack the CWF file, as each file is read as needed. Nonetheless, a C library (libcwf) is available that can read the CWF file format, for convenience.

The data for each region of the PicoTiterPlate device is stored in a separate CWF file. Because this file is fully self-contained, it is the only file that needs to be moved between the instrument and the data processing system for continued analysis (after the image processing phase of the GS Run Processor: this file format contains all the necessary information to generate any pipeline output artifact on-demand except for Standard Flowgram Format (SFF) files. Therefore, the user only needs to store one CWF file and one SFF file per region to fully archive the experiment's processed data.

Textual data in the CWF container will generally be stored as XML. Specifically, there are four main required XML streams: meta.xml, metrics.xml, history.xml, sequences.xml and one optional one, filters.xml. Each of these files references a single XML schema (which is available on request).

Table 5 shows an example listing of a CWF file's streams that might exist at the end of signal processing. This file represents the data from one region of a high-quality 2-region sequencing run on a GS FLX+ system, and the Table shows the size savings provided by the CWF compressed format. Each stream is described separately below.

Stream Name	**Uncompressed Size**	**Compressed Size**	**Compression Ratio**
mimetype	24	24	0%
rawWellDensity.pgm	8376341	1519187	82%
cfValues.double.dat	3749552	3328881	11%
ieValues.double.dat	3749552	3021477	19%
keyPassWellDensity.pgm	8376341	1103849	87%
histogram.unfiltered.ATGC.pgm	204695	24439	88%
histogram.unfiltered.TCAG.pgm	684743	40138	94%
filterResults.uint8.dat	468694	130450	72%
trimInfo.uint16.dat	937388	444081	53%
histogram.filteredCounts.ATGC.pgm	204695	22734	89%
histogram.nmers.ATGC.pgm	263879	20221	92%
histogram.filteredCounts.TCAG.pgm	684743	48430	93%
histogram.nmers.TCAG.pgm	1064508	60786	94%
0-149421.char.dna	33554213	10135865	70%
0-149421.uint_8.flow	33554213	9872137	71%
0-149421.uint_8.score	33554213	21064266	37%
149422-288120.char.dna	33554192	10070875	70%

Stream Name	Uncompressed Size	Compressed Size	Compression Ratio
149422-288120.uint_8.flow	33554192	9787599	71%
149422-288120.uint_8.score	33554192	20866664	38%
288121-424287.char.dna	33554389	10066101	70%
288121-424287.uint_8.flow	33554389	9787253	71%
288121-424287.uint_8.score	33554389	20863713	38%
424288-468693.char.dna	10323379	3139442	70%
424288-468693.uint_8.flow	10323379	3056298	70%
424288-468693.uint_8.score	10323379	6530831	37%
h0-41220.wel	33553894	29209536	13%
h41221-82441.wel	33553894	29206671	13%
h82442-123662.wel	33553894	29126347	13%
h123663-164883.wel	33553894	29069883	13%
h164884-206104.wel	33553894	29068170	13%
h206105-247325.wel	33553894	29082037	13%
h247326-288546.wel	33553894	29120018	13%
h288547-329767.wel	33553894	29184495	13%
h329768-370988.wel	33553894	29262051	13%
h370989-412209.wel	33553894	29361393	13%
h412210-453430.wel	33553894	29461534	12%
signalPerBase.float.dat	1874776	1539471	18%
h453431-468693.wel	12424082	10939973	12%
baseCalledSeq.dat	2812164	1126775	60%
sequences.xml	2334	612	74%
location.idx	4686940	2962553	37%
meta.xml	2162	701	68%
filters.xml	839	286	66%
metrics.xml	130111	18528	86%
history.xml	3286	1284	61%
TOTAL	752 MB	482MB	36%

Table 5: List of the streams in a CWF file following image and signal processing of a sequencing run.

2.3.1.1 mimetype

This is a single line file containing the words:

application/vnd.454.cwf

It must be the first stream in the file and must be stored uncompressed.

2.3.1.2 meta.xml

Like the OpenDocument format, meta.xml stores information about the run itself. As a convenience, the schema defining the XML data stored in the CWF file references the Dublin Core ("DC") metadata elements. The DC elements used and their interpreted meaning are summarized in Table 6. An example meta.xml file is shown in Figure 8.

DC Element	454 Sequencing System Usage
Title	Run name
Description	User-defined
Type	"flowgrams"
Source	Serial number of instrument performing the run
Relation	Original Run name. *e.g.* "R_2007_06_27_15_44_21_rig3_ccelone_1007075seqkit93555420PELTxxEX2xxVERIIF2"
Creator	Instrument operator's name
Date	"dcterms:created": Date analysis was performed
Identifier	UUID version of job

Table 6: Dublin Core metadata elements used in CWF files.

```
<?xml version="1.0" encoding="UTF-8"?>
<Metadata xmlns:tns="http://purl.org/dc/terms/"
xmlns:tnsa="http://purl.org/dc/elements/1.1/"
xmlns:tnsb="http://purl.org/dc/dcmitype/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="GSDataProcessing-1.0.xsd">
  <tnsa:title>R_2011_04_04_10_00_36_FLX08100646_Administrator_apr_2_build</tnsa:title>
  <tnsa:type>flowgrams</tnsa:type>
  <tnsa:creator>Administrator</tnsa:creator>
  <tns:created>2011-04-04T10:00:36Z</tns:created>
  <InstrumentSerialNumber>FLX08100646</InstrumentSerialNumber>
  <InstrumentVersion>2.6 (20110324_1135)</InstrumentVersion>
  <InstrumentModel>GSFLX</InstrumentModel>
  <InstrumentConfiguration>FLX+</InstrumentConfiguration>
  <Run>
    <Name>apr_2_build</Name>
    <RunId></RunId>
    <Project>Applications II</Project>
    <Kit>XL+KIT</Kit>
    <Script>400x_TACG_70x75_XLPLUSKIT.icl</Script>
    <RegionCount>2</RegionCount>
    <RegionLayoutName>2_region</RegionLayoutName>
    <PTP>
      <ID>749573</ID>
      <WellSize unit="um">35</WellSize>
      <Size unit="mm">
        <Width>70</Width>
        <Height>75</Height>
      </Size>
    </PTP>
    <Barcodes>
      <Barcode>123456</Barcode>
    </Barcodes>
    <Flow>
      <FlowCount>1603</FlowCount>
      <CycleCount>400</CycleCount>
      <ActualOrder>SSSSOOOOSOOOOSOOOOSOOOOSPSSSSTACG…TACGPS</ActualOrder>
      <FlowOrder>PTACG…TACGP</FlowOrder>
    </Flow>
    <Images>
      <ImageWidth>4096</ImageWidth>
      <ImageHeight>4096</ImageHeight>
      <DcOffset>495</DcOffset>
      <MaxValue>16383</MaxValue>
    </Images>
  </Run>
  <Region>
    <Name>region 1</Name>
    <Number>1</Number>
    <TemplateBounds unit="pixel">
      <Center>
        <X>1024</X>
        <Y>2048</Y>
      </Center>
      <Dimension>
        <Width>2047</Width>
        <Height>4095</Height>
      </Dimension>
    </TemplateBounds>
    <RevisedBounds unit="pixel">
      <Center>
        <X>1024</X>
        <Y>2048</Y>
      </Center>
      <Dimension>
        <Width>2047</Width>
        <Height>4095</Height>
      </Dimension>
    </RevisedBounds>
  </Region>
  <WellStatus>uncorrected</WellStatus>
  <WellCount>1104008</WellCount>
</Metadata>
```

Figure 8: Example meta.xml stream.

2.3.1.3 history.xml

This is a single XML file showing what processing has been done to these wells. It contains reference copies of the pipeline parameters that were used to create the final result, as well as processing times, dates, software revision numbers and a Universally Unique Identifier ("UUID") for each processing step. Each analysis performed on the data set adds a new "Job" element to the stream. An example history.xml file is shown in Figure 9.

```
<?xml version="1.0" encoding="UTF-8"?>
<History xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="GSDataProcessing-1.0.xsd">
  <Job>
    <ID>cc8dfc88-70ff-11e0-99ce-00215e70718c</ID>
    <Name>cwf_with_instrumentConfiguration_take3</Name>
    <ProcessingDirectoryName>D_2011_04_27_14_54_37_FLX08100646_imageProcessing
Only_cwf_with_instrumentConfiguration_take3</ProcessingDirectoryName>
    <OS>Linux</OS>
    <StartTime>2011-04-27T18:54:38Z</StartTime>
    <TotalJobSeconds>12435</TotalJobSeconds>
    <PartialJobSeconds>11927</PartialJobSeconds>
    <GsRunProcessorVersion>2.6</GsRunProcessorVersion>
    <GsRunProcessorBuild>20110426_1019</GsRunProcessorBuild>
    <Host>FLX08100646</Host>
    <NumDataSetsInJob>2</NumDataSetsInJob>
    <NumProcessors>4</NumProcessors>
    <PartialNumProcessors>2</PartialNumProcessors>
    <Type>imageProcessingOnly</Type>
    <Pipeline>imageProcessingOnly</Pipeline>
    <DisplayText>Image processing only</DisplayText>
    <Description>
            This is the default pipeline for taking the raw images and
            making intermediate files that can be analyzed by
            other pipelines.
        </Description>
    <CommandLine>
--run=/data/2011_04_04/R_2011_04_04_10_00_36_FLX08100646_Administrator
_apr_2_build/D_2011_04_27_14_54_37_FLX08100646_imageProcessingOnly_cwf_with_instrument
Configuration_take3/dataRunParams.xml
--imageLog=/data/2011_04_04/R_2011_04_04_10_00_36_FLX08100646_Administrator
_apr_2_build/imageLog.parse
--images=/data/2011_04_04/R_2011_04_04_10_00_36_FLX08100646_Administrator
_apr_2_build/rawImages/
--out=/data/2011_04_04/R_2011_04_04_10_00_36_FLX08100646_Administrator
_apr_2_build/D_2011_04_27_14_54_37_FLX08100646_imageProcessingOnly_cwf_with_instrument
Configuration_take3/regions
--log=/data/2011_04_04/R_2011_04_04_10_00_36_FLX08100646_Administrator
_apr_2_build/D_2011_04_27_14_54_37_FLX08100646_imageProcessingOnly_cwf_with_instrument
Configuration_take3/gsRunProcessor.log
--error=/data/2011_04_04/R_2011_04_04_10_00_36_FLX08100646_Administrator
_apr_2_build/D_2011_04_27_14_54_37_FLX08100646_imageProcessingOnly_cwf_with_instrument
Configuration_take3/gsRunProcessor_err.log
--job=cc8dfc88-70ff-11e0-99ce-00215e70718c
--pipe=/usr/local/rig/apps/gsRunProcessor/etc/gsRunProcessor/imageProcessingOnly.xml
--name=cwf_with_instrumentConfiguration_take3
--progress
--remoteProgress=localhost:4540
    </CommandLine>
    <ParamsUsed>
      <CameraClassifier computeTime="6" name="CameraClassifier">
        <enable>true</enable>
        <removeHotPixels>true</removeHotPixels>
        <imageScaleFactor>1</imageScaleFactor>
        <hotPixelThreshold>500</hotPixelThreshold>
        <computeSharpnessMask>true</computeSharpnessMask>
        <minPPI>30</minPPI>
        <standardDeviationLimit>2.2</standardDeviationLimit>
        <sampleBlockSize>128</sampleBlockSize>
        <fftRadius>45</fftRadius>
      </CameraClassifier>
      <WellFinder computeTime="398" name="WellFinder">
        <enable>true</enable>
        <imageScaleFactor>1</imageScaleFactor>
        <kernelSize>51</kernelSize>
        <upsampleHighDensityPtps>true</upsampleHighDensityPtps>
        <minPPISignal>30</minPPISignal>
        <minConsensusSignal>20</minConsensusSignal>
        <minWellSpacing>3</minWellSpacing>
        <secondSearchPass>false</secondSearchPass>
        <blockSize>20000</blockSize>
        <upsampleFactor>2</upsampleFactor>
        <maskBeta>0.6</maskBeta>
        <maskAlpha>0.1</maskAlpha>
        <numPixelsPerWell>1</numPixelsPerWell>
        <morphologyThresholdMultiplier>1</morphologyThresholdMultiplier>
        <morphologyNumInARow>5</morphologyNumInARow>
      </WellFinder>
      <WellBuilder computeTime="11520" name="WellBuilder">
        <enable>true</enable>
        <kernelSize>51</kernelSize>
        <minPPISignal>30</minPPISignal>
        <scaleFactor>1</scaleFactor>
        <useBicubic>true</useBicubic>
        <usePpiInterpolation>false</usePpiInterpolation>
        <firstFlowToInterpolate>20</firstFlowToInterpolate>
        <imageScaleFactor>1</imageScaleFactor>
        <skipBackgroundSubtraction>false</skipBackgroundSubtraction>
      </WellBuilder>
      <MetricsGenerator computeTime="1" name="MetricsGenerator">
        <enable>true</enable>
      </MetricsGenerator>
    </ParamsUsed>
    <FinishTime>2011-04-27T22:21:58Z</FinishTime>
    <TotalUserSeconds>37614</TotalUserSeconds>
    <TotalSystemSeconds>3974</TotalSystemSeconds>
  </Job>
</History>
```

Figure 9: Example history.xml stream.

2.3.1.4 location.idx

This is a binary index to the wells files. It contains common data about each well that can be used to support a well browser-style application. Items in this stream are stored in Intel Little-Endian format (*i.e.* the rank is stored as Byte3 Byte2 Byte1 Byte0). The wells file contains one field per well, and each field is made up of the packed structure shown in Figure 10:

	15															0
1	Rank (unsigned integer)															
2																
3																
4																
5	X Coordinate (integer part)															
6															fract.	
7	Y Coordinate (integer part)															
8															fract.	
9	P	Sequence ID														
10	Reserved															

Figure 10: Packed structure of the location.idx file.

- P is a bit showing if the well has passed filtering (1) or has been discarded (0).
- Sequence ID is an index to the table contained in sequences.xml (see below) showing which sequence (if any) is matched.
- The reserved field is for future use and should be set to all 0's.
- The X and Y coordinates are stored with two bits of fractional information. This allows the storage of well coordinates with sub-pixel resolution in the CWF file format. Applications accessing the coordinates may simply choose to map bytes 5/6 and 7/8 to 16-bit integers and divide by 4, preserving or discarding the subpixel data as needed. Also note that all coordinates are relative to a common "0,0" location representing the upper left hand corner of the PicoTiterPlate device (the corner opposite the DataMatrix code), no matter what the region. In other words, the coordinate reflects the actual location of the well on the original PicoTiterPlate device and not the offset into the region itself.

2.3.1.5 metrics.xml

This file contains all the derived statistics created during data processing. This file can be used to create all the ancillary output files, including all the reports that were produced by the 454 Sequencing system software versions anterior to 2.0.00. The data is divided into various sections in four main types.

- The first is the header section, which contains metrics that are valid across all keys and sequences.
- The next sections are the "MetricsPerKey," containing metrics that cover one key, either library or control. There will be one MetricsPerKey block per key used in the experiment.
- The next sections are the "MetricsPerSequence" blocks. The metrics for each Control DNA sequence are contained in separate blocks.
- The "Other" block is a free-form container for metrics that may be used by Roche and 454 Life Sciences' troubleshooters to evaluate problematic sequencing runs. These metrics can and will change between releases of software, and therefore, users should not depend upon them for the assessments of runs.

An additional block, the "Streams" block, acts as a manifest for data stored in auxiliary streams, inside the CWF file. Each data stream is individually tagged with a "type" identifier. The stream block contains the exact stream name, and the type of the binary data contained in the data. While the exact stream name and data type may change with future releases of the software, the "type" name will remain constant. Users implementing CWF file readers should use the streams information contained in metrics.xml file to find their data and not depend on a particular file naming convention. See the Sections on "Other Streams" (2.3.1.11 and 2.3.1.12) for information on the data types and stream names.

An example metrics.xml file is shown in Figure 11.

```
<?xml version="1.0" encoding="utf-8"?>
<Metrics xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="GSDataProcessing-1.0.xsd">
  <RunMetrics>
    <MaxWellCount>529820</MaxWellCount>
    <RawWellCount>468698</RawWellCount>
    <SampleKeyPassWellCount>451747</SampleKeyPassWellCount>
    <ControlKeyPassWellCount>7818</ControlKeyPassWellCount>
    <ControlKeys>
      <Key>ATGC</Key>
    </ControlKeys>
    <SampleKeys>
      <Key>TCAG</Key>
    </SampleKeys>
    <Streams>
      <Stream type="rawWellDensity">
        <StreamName>rawWellDensity.pgm</StreamName>
        <DataType>image</DataType>
      </Stream>
      <Stream type="carryForwardCorrections">
        <StreamName>cfValues.double.dat</StreamName>
        <DataType>double</DataType>
      </Stream>
      <Stream type="incompleteExtensionCorrections">
        <StreamName>ieValues.double.dat</StreamName>
        <DataType>double</DataType>
      </Stream>
        <Stream type="filterResults">
        <StreamName>filterResults.uint8.dat</StreamName>
        <DataType>byte</DataType>
      </Stream>
      <Stream type="signalPerBase">
        <StreamName>signalPerBase.float.dat</StreamName>
        <DataType>float</DataType>
      </Stream>
    </Streams>
    <Other>
      <NukeSignalStrengthBalancer>
        <medianOneMerA>1.09085</medianOneMerA>
        <medianOneMerT>0.909846</medianOneMerT>
        <medianOneMerG>0.898174</medianOneMerG>
        <medianOneMerC>0.894138</medianOneMerC>
      </NukeSignalStrengthBalancer>
      <BlowByCorrector>
        <droopLambda>-0.00171434</droopLambda>
        <MedianSignal>1375.71</MedianSignal>
        <MaximumSignal>5086.11</MaximumSignal>
        <MedianDensity>12</MedianDensity>
        <MinimumDensity>1</MinimumDensity>
        <MaximumDensity>19</MaximumDensity>
        <num_low_density_low_signal_wells>
        14742</num_low_density_low_signal_wells>
        <num_high_density_low_signal_wells>
        10481</num_high_density_low_signal_wells>
        <num_low_density_high_signal_wells>
        13645</num_low_density_high_signal_wells>
        <num_high_density_high_signal_wells>
        14899</num_high_density_high_signal_wells>
        <mask_averaging_used>true</mask_averaging_used>
        <FinalMask>
          <class density="high" signal="high" class="0">
            <epsilon>0.174537</epsilon>
            <beta>0.964658</beta>
          </class>
          <class density="low" signal="high" class="1">
            <epsilon>0.188713</epsilon>
            <beta>0.988837</beta>
          </class>
          <class density="high" signal="low" class="2">
            <epsilon>0.171184</epsilon>
            <beta>0.950913</beta>
          </class>
          <class density="low" signal="low" class="3">
            <epsilon>0.184087</epsilon>
            <beta>0.900547</beta>
          </class>
        </FinalMask>
      </BlowByCorrector>
      <CafieCorrector>
        <droopLambda>-0.00158241</droopLambda>
      </CafieCorrector>
      <NukeSignalStrengthBalancer>
        <medianOneMerA>1.01745</medianOneMerA>
        <medianOneMerT>0.986296</medianOneMerT>
        <medianOneMerG>0.987408</medianOneMerG>
        <medianOneMerC>0.980024</medianOneMerC>
      </NukeSignalStrengthBalancer>
    </Other>
  </RunMetrics>
</Metrics>
```

Figure 11: Example metrics.xml stream.

2.3.1.6 sequences.xml

A list of sequences referred to by the Sequence ID field in the locations.idx node. This stream can also be used to identify which sequences denote Control DNA reads and which are library reads. By convention, library keys are named “ATCG-library” where “ATGC” is the four letter library key. An example sequences.xml file is shown in Figure 12. (Note that the sequences were truncated (ellipses) on the Figure, for brevity).

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Sequences>
  <Sequence Type="None">
    <ID>0</ID>
    <Name>unknown</Name>
    <Key></Key>
    <Seq></Seq>
  </Sequence>
  <Sequence Type="Control">
    <ID>1</ID>
    <Name>ATGC-control</Name>
    <Key>ATGC</Key>
    <Seq>ATGC</Seq>
  </Sequence>
  <Sequence Type="Library">
    <ID>2</ID>
    <Name>TCAG-key</Name>
    <Key>TCAG</Key>
    <Seq>TCAG</Seq>
  </Sequence>
  <Sequence Type="Control">
    <ID>3</ID>
    <Name>TF2LonG</Name>
    <Key>ATGC</Key>
    <Seq>ATGCCA...TGTGTG</Seq>
  </Sequence>
  <Sequence Type="Control">
    <ID>4</ID>
    <Name>TF7LonG</Name>
    <Key>ATGC</Key>
    <Seq>ATGC...TTCCTGTGTG</Seq>
  </Sequence>
  <Sequence Type="Control">
    <ID>5</ID>
    <Name>TF90LonG</Name>
    <Key>ATGC</Key>
    <Seq>ATGCCGCA...GTGTG</Seq>
  </Sequence>
  <Sequence Type="Control">
    <ID>6</ID>
    <Name>TF100LonG</Name>
    <Key>ATGC</Key>
    <Seq>ATGCAT...GTGTG</Seq>
  </Sequence>
  <Sequence Type="Control">
```

```
    <ID>7</ID>
    <Name>TF120LonG</Name>
    <Key>ATGC</Key>
    <Seq>ATGCA...CCTGTGTG</Seq>
  </Sequence>
  <Sequence Type="Control">
    <ID>8</ID>
    <Name>TF150MMP7A</Name>
    <Key>ATGC</Key>
    <Seq>ATGCGC...ATGG</Seq>
  </Sequence>
</Sequences>
```

Figure 12: Example sequences.xml stream.

2.3.1.7 filters.xml

A list of filters referred to by the values in the "filterResults.uint8.dat" stream (see section 2.3.1.8, below). Note that the order of filters in this file is not guaranteed. It is also likely that the filters will be reorganized in a future release of the software to provide more detail. An example filters.xml file is shown in Figure 13.

```xml
<?xml version="1.0" encoding="utf-8"?>
<Filters xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="GSDataProcessing-1.0.xsd">
  <Filter basic="true">
    <ID>0</ID>
    <Name>Pass</Name>
  </Filter>
  <Filter basic="true">
    <ID>1</ID>
    <Name>No Key</Name>
  </Filter>
  <Filter basic="true">
    <ID>2</ID>
    <Name>Bad Band</Name>
  </Filter>
  <Filter basic="true">
    <ID>3</ID>
    <Name>Trimmed Too Short Quality</Name>
  </Filter>
  <Filter basic="true">
    <ID>4</ID>
    <Name>Low Pass Filter</Name>
  </Filter>
  <Filter basic="true">
    <ID>5</ID>
    <Name>Classifier Filter</Name>
  </Filter>
  <Filter basic="true">
    <ID>6</ID>
    <Name>Dot Filter</Name>
  </Filter>
  <Filter basic="true">
    <ID>7</ID>
    <Name>Mixed Filter</Name>
  </Filter>
  <Filter basic="true">
    <ID>8</ID>
    <Name>Trimmed Too Short Primer</Name>
  </Filter>
  <Filter basic="true">
    <ID>9</ID>
    <Name>Low Quality</Name>
  </Filter>
</Filters>
```

Figure 13: Example filters.xml stream.

2.3.1.8 filterResults.uint8.dat

This file contains a binary list of wells that failed filtering and the specific filter that the well did not pass. The filters are defined in file filters.xml. This stream consists of one byte per well, sorted by rank. See the discussion of "Other Streams" (sections 2.3.1.11 and 2.3.1.12) for a description on the layout of this stream.

2.3.1.9 Flow Data

The majority of the CWF payload consists of the well flow values themselves. In order to support rapid, random flow extraction from a CWF file, wells are stored in "blocks". The data format for the blocks can vary, depending on the application. The size of each block is equal to the largest multiple of the data size times the number of flows, that is smaller than 32 Megabytes. In other words, no well's flow values will be broken between blocks and no block will be larger than 32 MB.

The naming convention for each block of flowgrams is:

TY-Z.wel

… where

- T is a letter indicating the type of the block (see below),
- Y is the starting well index, and
- Z is the final well index.

Y and Z are integer values and should not be padded with zeros. The number of wells in each block is arbitrary and should not be hard coded. There is no assumption that all blocks in the file are stored in the same format nor that all blocks have the same number of wells. For example, Control DNA reads and failed wells may be archived in a lower fidelity format, while the library fragments are stored as full floating point numbers. However, the format cannot currently store discontinuous ranges of elements in a block, which constitutes a limitation of this feature. Users of the CWF format are encouraged to use libcwf to insulate them from the complexity of extracting the well information from the CWF file.

The block types are as follows:

- **r: Raw well block type**: This is identical to the *.wells format of software versions 1.1.03 and earlier. Values are stored "Little Endian" byte order as generated by Intel brand x86 processors.
- **Header**:
  - 32 bits: numWells (as unsigned integer)
  - 16 bits: numFlows (as unsigned integer)
  - numFlows bytes: flowLabels (one of "A","T","G","C","P")
- **Body**:
  - 32 bits: rank (as unsigned integer)
  - 16 bits: xCoord (as unsigned integer)
  - 16 bits: yCoord (as unsigned integer)
  - 32 bits * numFlows: flowValues (as IEEE Single Precision Float)
- **h: Half-precision floating point**: Each block is made up of four arrays stored back-to-back without padding. The size of the first three arrays is equal to the data type size times the number of wells in the block. The last array consumes the rest of the block, and is equal to 2 bytes times the number of wells times the number of flows. The number of wells is derived from the name of the stream, and the number of flows can be retrieved from the meta.xml stream. Values are stored "Little Endian" byte order as generated by Intel brand x86 processors.
  - X-coordinates as unsigned 16-bit numbers
  - Y-coordinates as unsigned 16-bit numbers
  - flow ranks as unsigned 32-bit numbers
  - flow values as "half-precision binary floating point numbers"

The half-precision floating point is a relatively new binary floating point format that uses 2 bytes and which is not covered by the IEEE 754 standard for encoding floating point numbers (but is included in the IEEE 754r proposed revision; http://www.validlab.com/754R/). The format uses 1 sign bit, a 5-bit excess-15 exponent, 10 mantissa bits (with an implied 1 bit) and all the standard IEEE rules. The minimum and maximum representable values are 2.98×10^{-8} and 65504, respectively. Libcwf includes a half to full precision floating point conversion routine.

- **f: Full-precision floating point**: Each block is made up of four arrays stored back-to-back without padding. The size of the first three arrays is equal to the data type size times the number of wells in the block. The last array consumes the rest of the block, and is equal to 4 bytes times the number of wells times the number of flows. The number of wells is derived from the name of the stream, and the number of flows can be retrieved from the meta.xml stream. Values are stored "Little Endian" byte order as generated by Intel brand x86 processors.
    - X-coordinates as unsigned 16-bit numbers
    - Y-coordinates as unsigned 16-bit numbers
    - flow ranks as unsigned 32-bit numbers
    - flow values as IEE 754 binary full precision floating point number (http://grouper.ieee.org/groups/754/)
- **i: Integer**: Each block is made up of four arrays stored back-to-back without padding. The size of the first three arrays is equal to the data type size times the number of wells in the block. The last array consumes the rest of the block, and is equal to 2 bytes times the number of wells times the number of flows. The number of wells is derived from the name of the stream, and the number of flows can be retrieved from the meta.xml stream. Values are stored "Little Endian" byte order as generated by Intel brand x86 processors. (Note that this block is rarely used because it cannot store values less than zero, which can occur during signal processing routines, and lacks precision for values near one.)
    - X-coordinates as unsigned 16-bit numbers
    - Y-coordinates as unsigned 16-bit numbers
    - flow ranks as unsigned 32-bit numbers
    - flow values as IEE 754 binary full-precision floating point number

2.3.1.10 Base Called Data

If a data set has processed through the baseCaller section of the GS Run Processor, the actual bases are written into the CWF file. This allows for the generation of FASTA FNA and QUAL files on demand. Like the flowgrams, the reads are stored in blocks. Unlike the flowgrams, however, each block consists of a variable number of reads. A special stream called "baseCalledSeq.dat" is used to index the basecalled blocks. This stream contains 6 bytes per well:

- Byte 0/1: Total stored read length
- Byte 2/3: Number of reads to trim from the distal end (3')
- Byte 4/5: Number of reads to trim from the local end (5')

The complete data for any read is stored in three separate streams, identified by the "dna", "qual" and "flow" extensions, containing the reads, the PHRED based quality values and the offset flow indices, respectively. Each base of each well uses one byte in each file.

The total number of bytes consumed by a read is reflected in the first field of the baseCalledSeq.dat. Therefore these two bytes can be used as an index of sorts. For example, the byte offset in the "dna" file for read 100 can be found by summing the first two bytes of the first 99 entries in baseCalledSeq.dat. The 100th entry can then be used to tell how many bytes are available for read 100. It is important to note that since the basecalls are stored in blocks, one must first find the appropriate block, then compute the offset from there. Again, users of the CWF format are encouraged to use the libcwf to insulate them from errors in extracting the base information.

2.3.1.11 Other Public Streams

The cwf file can also contain other binary streams. These are usually identified by the .dat suffix (except in the case of the "image" type, see note below). Many are used for storing intermediate results of various processing stages, but there are three notable streams that contain metrics data that may be of interest to end users.

- The first, "filterResults," can be used to find which wells have passed which filter.
- The second, "rawWellDensity" is a grayscale image containing the bead loading density plot.
- The last is the "keyPassDensity" stream, a grayscale image showing the keypass density.

With the exception of the "image" format stream, each stream contains one entry per read, sorted in rank order. The location.idx can be used to find which offset corresponds to which read.

The possible data types are listed in Table 7, and the current stream types, in Table 8.

Name	Size in Bytes/Well	Notes
byte	1	
short	2	Signed short, Intel byte order
unsignedShort	2	Unsigned short, Intel byte order
int	4	Standard integer, intel byte order
unsignedInt	4	Unsigned integer, Intel byte order
float	4	IEEE 754 Single Precision floating point number
double	4	IEEE 754 Double Precision floating point number
image	N/A	PGM format graphics. See note below on resolution/ registration.

Table 7: Possible data types.

Stream Type	Contents
filterResults	The information about which well failed which filter in the qualityFiltering section of code.
trimInfo	The trim points from the end of the reads. This number is in flowgram-space instead of base-space like the information in the baseCallerSeq.dat stream.
cfValues	The carry-forward corrections for each well.
ieValues	The incomplete extension correction factors for each well.
signalPerBase	The average value of the keypass flows for this well. Can be used to do a simple basecalling. Also can be used to judge the strength of the well.
keyPassDensity	An image referenced to the region showing the density wells that pass key. This is normalized for each PTP pitch, so 0=0% loading and 255=100% loading.
rawWellDensity	An image referenced to the region showing the density of the bead loading. This is normalized for each PTP pitch, so 0=0% loading and 255=100% loading.

Table 8: Current stream types.

Note on Image Formats: The only image format stored in CWF is the lossless Portable Anymap format, specifically the "P5" PGM (Portable Graymap) variant (http://netpbm.sourceforge.net/doc/pgm.html). To save space, only the area encompassed by the region is included in the image. To properly register the image against the original PTP device, you must offset the image slice by the region boundary. This region boundary can be read in the "RevisedBounds" element of the "Region" block in meta.xml stream. 454 Life Sciences Corporation may introduce other image formats in future variants of the CWF file, so it is important to read the "magic number" and/or file extension of the image to determine the correct image decoder to use.

2.3.1.12 Other Private Streams

There are other streams of data that can be stored in the CWF file, that contain intermediate results from various pipeline and post-processing functions. These are left unspecified to avoid restricting the development of new algorithms in the data processing applications. CWF file users should neither depend on their existence nor attempt to parse them as their contents may change between revisions of the software or invocations of different processing streams.

2.3.1.13 Other Files

There may be other files in the CWF container, the contents of which may include log files and other binary data. A proper CWF writer (like the GS Run Processor) will copy any other unknown stream verbatim to the destination. CWF file readers should not balk at these extra streams, but should not depend on their existence either. This allows advanced users to add additional payloads to the CWF container before moving them to the data analysis system.

2.3.2 Parameters and Viewable Metrics Files: 454 Parser Format (.parse, .txt)

The "parser" format, developed at 454 Life Sciences Corporation, is a standard format used for all the software parameter files and for most of the metrics files. This is a text-based format that organizes the text in titled "groups" that contain either sub-groups or name/value pairs of strings. A parser file consists of one or more of the following:

- A C-style comment, using /* and */ to delineate the comment text
- A group, whose syntax is '`groupname { … }`' and where one or more comments, sub-groups or name/value pairs can occur between the braces
- A name/value pair, whose syntax is '`name = "value";`' and where the quotes around the value are optional, but the equals sign and semi-colon are required.

The parser file format is free-form, *i.e.* the syntactic elements can appear with any style of white space or line division. However, the output files generated by the 454 Sequencing system software use a standard indentation convention where the group names appear by themselves on a line with the braces below it; all the text between the braces is indented; and each name/value pair appears on a single line.

Several examples of parser files are shown in the Output sub-sections of the various applications' descriptions.

2.3.3 Image Files (.pif and .png)

Image data from the GS FLX+ Instrument are stored as compressed industry-standard .png image files (http://www.w3.org/TR/PNG/).

The PNG format allows metadata to be included in each image file for ease of access. Currently, the image data from the GS FLX+ Instrument replicates the following annotations found in the imageLog.parse file.

PNG File Annotation	Example
Title	:454 Genome Sequencer camera image
Description	:Image X of Y for flow apyrase for run R_xxx
Creation Time	:24 01 2011 20:51:26 GMT
Software	:2.6 (20110120_1556)
Source	:camera model 806 serial 591
Host	:FLX08100646
Run	:R_2011_01_24_15_24_20_FLX08100646_A…
Exposures	:1634
Exposure	:11
Flow	:apyrase08
Exposure Time (ms)	:36000
Exposure Complete Time (sec)	:1295902285.041494
DC Offset	:495
Camera Model	:806
Camera Serial	:591
CCD Temperature (C)	:-025.05
PTP ID	:749558

Table 9: PNG File Metadata.

The legacy .pif image format is still supported, so older run image data (and data from GS Junior Instrument runs) can be used directly, without re-processing. The header is 12 bytes long, comprised of three 4-byte integers: the first integer value is the number of bits per pixel of data; the second integer is the width of the image in pixels; and the third integer is the height of the image in pixels. The following data in the file are the pixel intensity values, presented in row major order starting in the upper left corner. Currently, all image data is stored in 16 bit unsigned integers, or 2 bytes per pixel. Valid image data is limited to the first 14 bits.

2.3.4 Well-Level Signal Data Files (.wells)

The wells data file is a legacy binary file containing counts values for the light collected on the nucleotide and PPi images, at each "active" well location. The file consists of a header and a body, where the header contains the following fields:

```
unsigned int numWells;
unsigned short numFlows;
char flowOrder[numFlows];
```

…where numWells is the number of wells (or reads) in the file, numFlows is the number of flows in the sequencing run script, and flowOrder are characters indicating the reagent for each flow: 'A', 'C', 'G' and 'T' specify each nucleotide flow, and 'P' signifies a PPi flow.

The body of the wells data file contains numWells records of the following fields:

```
unsigned int rank;
unsigned short x;
unsigned short y;
float flowValues[numFlows];
```

…where rank is the general ranking of the well (by signal intensity); x and y are the coordinates of the well center pixel; and flowValues are the signal values for all the flows in this well.

All the multi-byte values in the header and body are written using little endian byte ordering (consistent with the Linux operating system). As these are binary-format files, an example cannot be provided in this text-based document.

2.3.5 Exportable Metrics Files (.csv)

The comma-separated values text file (.csv) is an alternate format in which a number of the metrics files are output. This format is suitable for automated parsing by programs or for loading into a spreadsheet program like Microsoft Excel. The content of these files, where generated, is identical to the corresponding file formatted in the 454 parser format described above (.txt).

2.3.6 DNA Sequence (FASTA; .fna) and Base Quality Score (.qual) Files

Three of the 454 Sequencing system data processing applications output DNA sequences: Signal Processing, for the basecalls of individual reads; GS *De Novo* Assembler, for the *de novo*-assembled consensus sequence of the sample DNA library; and GS Reference Mapper, for the sample's consensus sequence mapped to a reference sequence. These use the FASTA standard file format (.fna), and are always accompanied by a corresponding base quality scores file, in the .qual format. Examples are shown in the Output sub-sections of these applications' descriptions (*e.g. region.key*.454Reads and 454AllContigs).

Upper case and lower case letters in DNA sequences represent one of two distinct characteristics; either trim status or base quality. Lower case letters in raw (untrimmed) reads are used to represent bases outside of the trim points, including both the sequencing key at the 5' end and the bases trimmed from the 3' end during signal processing. In contigs, lower case letters are used to represent bases with a consensus basecall accuracy of <99.99% (Q<40).

Note that the description lines are slightly different depending on whether the FASTA file outputs contain reads or contigs, and for contigs, whether they were generated by the GS *De Novo* Assembler or by the GS Reference Mapper application.

1. For individual reads, the description lines are formatted as:

   >*rank_x_y* length=*XX*bp uaccno=*accession*

   …where "*rank_x_y*" is the identifier or accession number of the read (the rank, x and y values are as described in section 2.3.4), *XX*bp is the length in bases of the read, and *accession* is the full universal accession number for the read.
2. For contigs generated by the GS *De Novo* Assembler application, the description lines are formatted as follows:

   >contig*XXXXX* length=*abc* numReads=*xyz*

   …where "contig*XXXXX*" is the identifier of the contig and "*XXXXX*" is a sequential numbering of the contigs in the assembly; and where the length and numReads values are the length in bases of the contig and the number of reads that were used in that contig's multiple alignment.
3. For contigs generated by the GS Reference Mapper application, the description lines are formatted as follows:

   >contig*XXXXX* *refaccno*, *YYY..ZZZ* length=*abc* numReads=*xyz*

   …where "contig*XXXXX*" is the identifier of the contig and "*XXXXX*" is a sequential numbering of the contigs along the reference; "*refaccno*" is the accession of the reference sequence where this contig aligns; "*YYY..ZZZ*" is the start and end position of the contig on that reference sequence; and the length and numReads values are the length in bases of the contig and the number of reads that were used in that contig's multiple alignment.

2.3.7 Nucleotide Alignment (BAM; .bam) Files

GS Reference Mapper can optionally output sequence alignment data in BAM format, which can be loaded by a variety of third-party alignment viewer and variant calling programs. BAM files are the binary equivalent of the Sequence Alignment/Map (SAM) format file, compressed using the BGZF compression format. The SAM and BAM format specifications are described in detail at http://samtools.sourceforge.net/SAM1.pdf.

2.3.8 454 Sequencing System "Universal" Accession Numbers

To allow for the unique identification of reads across larger data sets, a unique accession number format has been developed. An accession in this format is a 14 character string, as in C3U5GWL01CBXT2, and consists of 4 components:

C3U5GW	- a six character encoding of the timestamp of the run
L	- a randomizing "hash" character to enhance uniqueness
01	- the region the read came from, as a two-digit number
CBXT2	- a five character encoding of the X,Y location of the well

The timestamp, hash character and X,Y location use a base-36 encoding (where values 0-25 are the letters 'A'-'Z' and the values 26-35 are the digits '0'-'9'). An accession thus consists only of letters and digits, and is case-insensitive.

- The timestamp is encoded by computing a "total" value as shown below, then converting it into a base-36 string:

```
total =
        (year - 2000) * 13 * 32 * 24 * 60 * 60 +
        month * 32 * 24 * 60 * 60 +
        day * 24 * 60 * 60 +
        hour * 60 * 60 +
        minute * 60 +
        second;
```

As a result of this calculation, the first character of read accessions will always be a letter for runs performed from now until 2038. The timestamp values are taken from the rigRunName found in the analysisParms.parse file in the specified analysis directory. This rigRunName is the R_... name that is generated by the instrument software, and is also used as the standard directory name for the run. Thus, a run whose name begins with R_2004_09_22_16_59_10_... generates C3U5GW as its encoded timestamp value.

- Since two runs may be started at the same second, an additional base-36 character is generated by hashing the full rigRunName to a base-31 number (the highest prime below 36), as in:

```
chval = 0;
for (s=rigRunName; *s; s++) {
   chval += (int) *s;
   chval %= 31;
}
ch = (chval < 26 ? 'A' + chval : '0' + chval - 26);
```

- The X,Y location is encoded by computing a total value of "X * 4096 + Y" and encoding that as a five character, base-36 string.

2.3.9 Standard Flowgram Files (.sff)

The Standard Flowgram File is used to store the information on one or many 454 Sequencing reads and their trace data. Sequencing reads obtained using the 454 Sequencing system differ from reads obtained using more traditional methods ("Sanger sequencing") in that the 454 Sequencing data do not provide individual base measurements from which basecalls can be derived. Instead, it provides measurements that estimate the length of each homopolymer stretch in the sequence (*e.g.* in "AAATGG", "AAA" is a 3-mer stretch of A, "T" is a 1-mer stretch of T and "GG" is a 2-mer stretch of G). A basecalled sequence is then derived by converting each estimate into a homopolymer stretch of that length and concatenating the homopolymers.

Use sfffile (see Part C of this manual) to manipulate the content of .sff files, for example to extract subsets of individual reads, or to merge multiple read files that share the same flow pattern and flow list.

Use sffinfo (see Part C of this manual) to view the content of a binary (non-readable) .sff file as plain text (the –S option turns off word wrapping at the end of the line).

```
sffinfo readfile.sff | less –S
```

The .sff file format consists of three sections: a common header section occurring once in the file; then for each read stored in the file, a read header section and a read data section. The data in each section consists of a combination of numeric and character data; the specific fields for each section are defined below. The sections adhere to the following rules:

- The standard Unix types uint8_t, uint16_t, uint32_t and uint64_t are used to define 1, 2, 4 and 8 byte numeric values.
- All multi-byte numeric values are stored using big endian byteorder (same as the SCF file format).
- All character fields use single-byte ASCII characters.
- Each section definition ends with an "eight_byte_padding" field, which consists of 0 to 7 bytes of padding, so that the byte length of each section is divisible by 8 (and hence the next section is aligned on an 8-byte boundary).

2.3.9.1 Common Header Section

The common header section consists of the fields displayed in the following table, and can be viewed using sffinfo (see Part C of this manual):

```
sffinfo -c readfile.sff
```

Field name	Format	Properties
magic_number	uint32_t	The magic_number field value is 0x2E736666, the uint32_t encoding of the string ".sff".
version	char[4]	The version number is 0001, or the byte array "\0\0\0\1".
index_offset	uint64_t	The index_offset and index_length fields are the offset and length of an optional index of the reads in the SFF file. If no index is included in the file, both fields must be 0.
index_length	uint32_t	
number_of_reads	uint32_t	The number_of_reads field should be set to the number of reads stored in the file.
header_length	uint16_t	The header_length field should be the total number of bytes required by this set of header fields, and should be equal to "31 + number_of_flows_per_read + key_length", rounded up to the next value divisible by 8.
key_length	uint16_t	The key_length field should be set to the length of the key sequence used for these reads.
number_of_flows_per_read	uint16_t	The number_of_flows_per_read should be set to the number of flows for each of the reads in the file.
flowgram_format_ code	uint8_t	The flowgram_format_code should be set to the format used to encode each of the flowgram values for each read. Currently, only one flowgram format has been adopted, so this value should be set to 1. The flowgram format code 1 stores each value as a uint16_t, where the floating point flowgram value is encoded as "(int) round(value * 100.0)", and decoded as "(storedvalue / 100.0)". In other words, the values are stored as an integer encoding of a limited precision floating point value, keeping 2 places to the right of the decimal point, and capping the values at 655.35.
flow_chars	char[number_of_ flows_per_read]	The flow_chars should be set to the array of nucleotide bases ('A', 'C', 'G' or 'T') that correspond to the nucleotides used for each flow of each read. The length of the array should equal number_of_flows_per_read. Note that the flow_chars field is not null-terminated.
key_sequence	char[key_length]	The key_sequence field should be set to the nucleotide bases of the key sequence used for these reads. Note that the key_sequence field is not null-terminated.
eight_byte_ padding	uint8_t[*]	If any eight_byte_padding bytes exist in the section, they should have a byte value of 0.

If an index is included in the file, the index_offset and index_length values in the common header should point to the section of the file containing the index. To support different indexing methods, the index section should begin with the following two fields:

index_magic_number	uint32_t
index_version	char[4]

… and should end with an eight_byte_padding field, so that the length of the index section is divisible by 8. The format of the rest of the index section is specific to the indexing method used. The index_length given in the common header should include the bytes of these fields and the padding.

2.3.9.2 Read Header Section

The rest of the file contains the information about the reads, namely number_of_reads entries consisting of read header and read data sections. Each read header section consists of the following fields:

Field name	Format	Properties
read_header_length	uint16_t	The read_header_length should be set to the length of the read header for this read, and should be equal to "16 + name_length" rounded up to the next value divisible by 8.
name_length	uint16_t	The name_length field should be set to the length of the read's accession or name.
number_of_bases	uint32_t	The number_of_bases should be set to the number of bases called for this read.
clip_qual_left	uint16_t	The clip_qual_left and clip_adapter_left fields should be set to the position of the first base after the clipping point, for quality and/or an adapter sequence, at the beginning of the read. If only a combined clipping position is computed, it should be stored in clip_qual_left. The clip_qual_right and clip_adapter_right fields should be set to the position of the last base before the clipping point, for quality and/or an adapter sequence, at the end of the read. If only a combined clipping position is computed, it should be stored in clip_qual_right. Note that the position values use 1-based indexing, so the first base is at position 1. If a clipping value is not computed, the field should be set to 0. Thus, the first base of the insert is: max(1, max(clip_qual_left, clip_adapter_left)) …and the last base of the insert is: min((clip_qual_right == 0 ? number_of_bases : clip_qual_right), (clip_adapter_right == 0 ? number_of_bases : clip_adapter_right))
clip_qual_right	uint16_t	
clip_adapter_left	uint16_t	
clip_adapter_right	uint16_t	
Name	char[name_length]	The name field should be set to the string of the read's accession or name. Note that the name field is not null-terminated.
eight_byte_padding	uint8_t[*]	If any eight_byte_padding bytes exist in the section, they should have a byte value of 0.

2.3.9.3 Read Data Section

The read data section consists of the following fields:

Field name	**Format**	**Properties**
flowgram_values	uint*_t[number_of_flows]	The flowgram_values field contains the homopolymer stretch estimates for each flow of the read. The number of bytes used for each value depends on the common header flowgram_format_code value (where the current value uses a uint16_t for each value).
flow_index_per_base	uint8_t[number_of_bases]	The flow_index_per_base field contains the flow positions for each base in the called sequence (*i.e.*, for each base, the position in the flowgram whose estimate resulted in that base being called). Note that these values are "incremental" values, *i.e.* the stored position is the offset from the previous flow index in the field. All position values (prior to their incremental encoding) use 1-based indexing, so the first flow is flow 1.
Bases	char[number_of_ bases]	The bases field contains the basecalled nucleotide sequence.
quality_scores	uint8_t[number_of_bases]	The quality_scores field contains the quality scores for each of the bases in the sequence, where the values use the standard -log10 probability scale.
eight_byte_padding	uint8_t[*]	If any eight_byte_padding bytes exist in the section, they should have a byte value of 0.

2.3.9.4 Computing Lengths and Scanning the File

The length of the various read's section will vary because of different length accession numbers and different length nucleotide sequences. However, the various flow, name and bases lengths given in the common and read headers can be used to scan the file, accessing each read's information or skipping read sections in the file. The following pseudocode gives an example method to scan the file and access each read's data:

```
1.  Open the file and/or reset the file pointer position to the first byte of the file.
2.  Read the first 31 bytes of the file, confirm the magic_number value and version, then extract the number_of_reads,
    number_of_flows_per_read, flowgram_format_code, header_length, key_length, index_offset and index_length values.
    a.  Convert the flowgram_format_code into a flowgram_bytes_per_flow value (currently with format_code 1, this
        value is 2 bytes).
3.  If the flow_chars and key_sequence information is required, read the next "header_length - 31" bytes, then extract that
    information. Otherwise, set the file pointer position to byte header_length.
4.  While the file contains more bytes, do the following:
    a.  If the file pointer position equals index_offset, either read or skip index_length bytes in the file, processing the
        index if read.
    b.  Otherwise,
        i.   Read 16 bytes and extract the read_header_length, name_length and number_of_bases values.
        ii.  Read the next "read_header_length - 16" bytes to read the name.
        iii. At this point, a test of the name field can be performed, to determine whether to read or skip this entry.
             (a)  Compute the read_data_length as

             "number_of_flows * flowgram_bytes_per_flow + 3 * number_of_bases"

             …rounded up to the next value divisible by 8.

             (b)  Either read or skip read_data_length bytes in the file, processing the read data if the section is read.
```

Published by
454 Life Sciences Corp.
A Roche Company
Branford, CT 06405
USA

(9) 0613