# InterOp Parsers

To obtain the required data from the raw files, parsers were written in the Python programming language (version 2.7.11, 64bit). Development utilised the PyDev plugin (release 4.5.4) for the Eclipse IDE (integrated development environment) (version 4.4.2) on a PC running the Microsoft Windows 10 operating system(118–120). This programming language, development environment and computer were used for the entire project. Originally the parsers were written as procedural programs, as part of the initial development cycle to explore the structure of the binary files and possible methods for accessing the information, including locating and testing potentially useful existing libraries (Appendix 1). Subsequently, a modular approach to code design was taken wherever possible to facilitate troubleshooting, maintenance and upgrading. Git (open source software) was used for version control, with files regularly pushed to GitHub for back-up and potential future collaboration. A link to this GitHub repository is in Appendix 1(121–123).

The scripts and a brief explanation of their function is outlined in Table 2.2. An overview of the structure of the code is in Figure 2.2. Import.py is the script which is run by the user. In the future it is hoped to automatically trigger this script after a sequencing run has completed, but this is not yet implemented. Import.py iterates through all of the folders present in a manually specified directory (currently set to the location of the subset of raw data files obtained from the cluster archive location as described above), and iterates through files present within that directory. Files which match the required subset of files (runParameters.xml, SampleSheet.csv and the binary files present within the InterOp folder) are loaded and parsed by other scripts, called as classes by Import.py.

Table 1: Parsing scripts

|  |  |
| --- | --- |
| Script | Purpose |
| Import.py | Iterates through folders within a specified directory and calls other functions. Prints out the name of the folder so that progress can be tracked. |
| CheckExists.py | A class containing a function that checks if there is already an entry in the database matching the run unique identifier. |
| PopulateTablesClass.py | A class containing a series of functions to extract the required data from the runParameters.xml and SampleSheet.csv files and insert it into the database. |
| RunParametersParser.py | A class containing a series of functions that extract the required data from the runParameters.xml file. |
| RunParamsUnix.py | A class containing a function to check the existence and then the case of the runParameters.xml file and ensure that the script calling it has the correct case. This was created as it was previously noted that this file could be in one of two different cases (runParameters.xml or RunParameters.xml depending on kit version used). This is not an issue when running the scripts on a Windows operating system as it is generally case insensitive, however, Unix-based operating systems, such as that on the cluster, are case sensitive. |
| SampleSheetCheck.py | A class containing a function to check if the SampleSheet.csv file is present. The current version returns the sample sheet name if a sample sheet is present, otherwise it returns ‘None’. The scripts will enter all the data except that derived from the sample sheet if there is no sample sheet (such as in the case of the data provided by the University). |

|  |  |
| --- | --- |
| SampleSheetParser.py | A class containing a series of functions that extract required data from the SampleSheet.csv file. |
| PopulateLinkMSRRds.py | A class containing a function to populate the table linking the miseqrun table and the reads table. |
| PopulateBinaryTablesClass.py | A class containing a function to control extraction of data from the binary files in the InterOp folder which follow a common format (not the ExtractionMetrics.bin or the IndexMetrics.bin that follow a different format and have separate classes (see below)) and the runParameters.xml file, and population of the database with this data(105). |
| ParseInterOpMetrics.py | A class containing a series of functions to extract data from the binary InterOp files. These functions are called as appropriate from other scripts to return a required subset of data. |
| PopulateBinaryTablesExtractionClass.py | A class containing a function to control extraction of data from the ExtractionMetrics.bin binary file in the InterOp folder and the runParameters.xml file, and population of the database with this data(105). |
| PopulateBinaryTablesIndexClass.py | A class containing a function to extract data from the IndexMetrics.bin binary file in the InterOp folder and the runParameters.xml file, and population of the database with this data(105). |

Table 1: The scripts used to parse the raw data file to retrieve the required subset of data. Earlier versions of the scripts printed out the extracted data enabling it to be checked that the scripts ran and the output matched the expected values obtained by inspection in the SAV software(105). This version of the scripts also populates the database with the data.

Figure 1 Overview of the parsing scripts

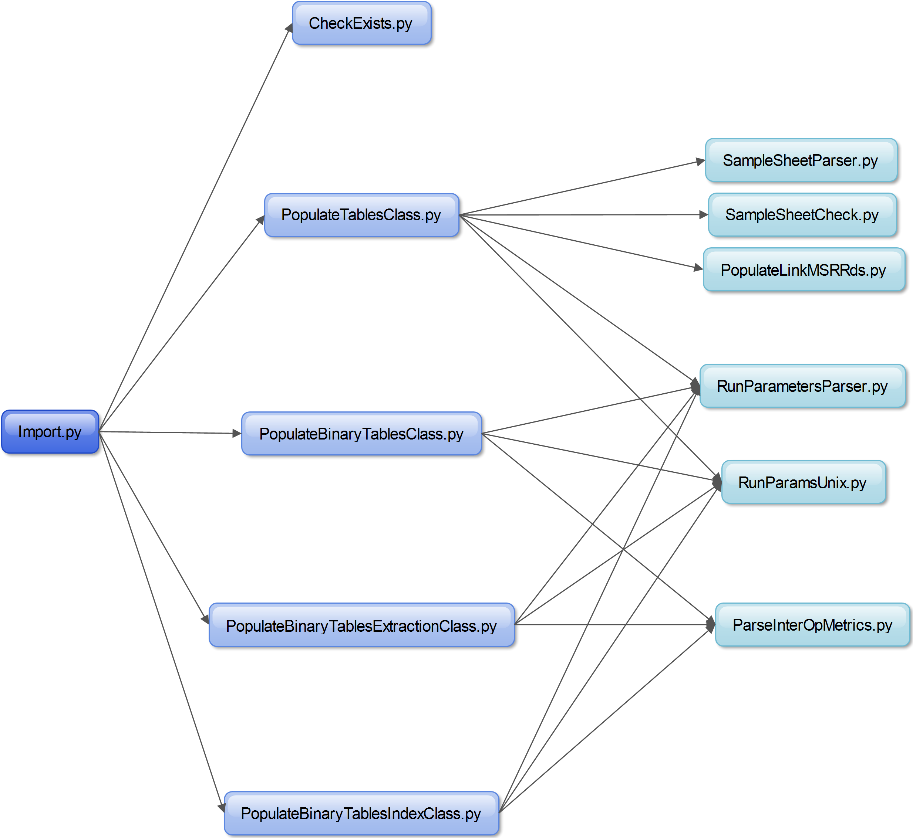


Figure 1: The script Import.py is run by the user and calls the scripts in the second column. These scripts then call the scripts in the third column to which they are linked by arrows. All of the scripts in the second column call RunParametersParser.py and RunParamsUnix.py with the exception of CheckExists.py, which calls no other scripts.

#### Rationale for creating custom parsers

There are existing methods to inspect and explore data contained within the InterOp binary files, however none of these methods met the requirements for this project. These methods and the reasons they were considered unsuitable for use in this project are discussed below.

As previously mentioned, data from the InterOp binary files can be inspected and manipulated through an application provided by Illumina called Sequencing Analysis Viewer (SAV). This software provides an easily accessible, largely graphical summary of the important quality metrics generated by the sequencing instrument (Figure 2.3). In addition to documentation and guides, training is provided by Illumina in the form of webinars to enable individual users to understand and interpret the data presented in this program and make assessments about the quality of their sequencing(105,107). While this application is extremely useful in quality control, and would circumvent the need to create parsers for the InterOp binary data, it is not suitable for the goals of this project as it is designed for human-interaction and raw data is not readily accessible, although some data can be manually exported through the graphical user interface (as a .csv file) from the “Imaging” tab of the application.

Figure 2 Screenshot of the Illumina SAV application



Figure 2: A screenshot of the Analysis tab from the Illumina SAV software. This tab is manually inspected for the majority of the QC checks performed by sequencing operators, although informative data is also found in the Summary (cluster density) and Indexing (relative proportion of reads per sample) tabs.

This data from the “Imaging” tab can also be extracted by running the SAV from the Microsoft Windows command line with the -t option specified as the third command line argument (after the path to the SAV and the path to the run data from which data is to be extracted). A csv file is output that contains some metrics derived from the binary InterOp files and a useful summary of some of the data (Figure 2.4). Initially it was thought that it may be possible to use the SAV to obtain this file and then parse it to populate the QC database, negating the need to develop custom InterOp file parsers. However, the data in this table is incomplete, for example there is no information regarding the index metrics. Furthermore, this method of using the application is unsupported by Illumina (there is no reference to it in any documentation) and subsequently may stop working at any time. Finally, usage of the SAV software requires a Microsoft Windows operating system and cross-platform compatibility is a requirement of the project. Therefore, it was considered necessary to continue to explore other methods of extracting data from the InterOp binary files for the purposes of this project.

Figure 3 Table of data in SAV

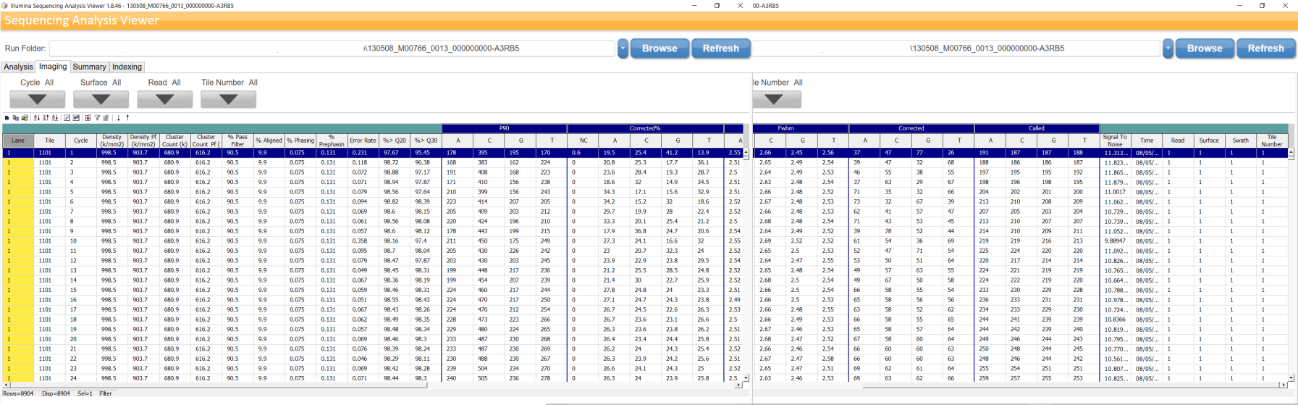


Figure 3: The table of data available within the SAV application that can also be extracted from within this software through a graphical user interface(105).

This method of extracting data from the SAV does, however, offer a useful basis for comparison with the output of custom parsers. It provides a source of data known to be correct, as the application has been developed and is supported by the manufacturer of the sequencing instrument(105). As manually opening each individual run and extracting the data from within the SAV application would be time-consuming, requiring multiple steps, a batch script (Appendix 1) was written to extract this table as a comma separated file, a readily accessible (easily openable with spreadsheet programs) format.

There is an existing Python package available called illuminate that allows parsing of Illumina generated binary files and outputs the results in the Python dictionary format(125). As this package is in the Python language and also supports integration with other software it would have been beneficial to the project to be able to use this package rather than spend time on development of custom parsers. However, the package requires a UNIX-like environment to run(125). It is therefore not cross-platform compatible and so does not meet the requirements of this project.

As, for the reasons outlined above, the identified potential solutions were unsuitable for this project, it was considered necessary to develop parsers for accessing the information encoded in the InterOp binary files.

#### Development of the parsers, major challenges and limitations

An agile approach was taken towards development, with code written and then tested using a single dataset after every change(126). This enabled bugs and issues, ranging from issues with unpacking of the binary files leading to incorrect values being obtained, to immediate exceptions being thrown by the built-in error handlers, to be immediately identified and fixed. The values obtained from the SAV with the same dataset were used as a truth set for comparison.

The struct module of the Python standard library was used to unpack the binary data and information regarding the structure of the InterOp files was provided by Illumina, however, unpacking the binary files to yield the correct results was more challenging than anticipated(105,127). Firstly, during the process of development of the parsers, it was noticed that the specification document provided by Illumina contained an error in the number of bytes encoding the intensities for each base. This was specified as 2x4 bytes, but was actually 4x2 bytes(105). The source of the resultant error with the ExtractionMetricsOut.bin parsing took some time to identify as it was initially assumed that the specifications were correct and there was an issue with the implementation. Secondly, some areas required extensive work in terms of planning and implementation of suitable methods to unpack some of the information. Unpacking the date-time field of the ExtractionMetricsOut.bin file, required converting a 64-bit integer (8 bytes) into binary, replacement of the first two bits of this binary number with 0s (as in .NET encoding these are used to store “kind”), reversing of the binary number to match little-endian encoding (file created on a Windows machine), converting the binary number back into decimal format and then converting the “ticks” (the time since 12:00:00 on 1st January 0001 in 100 nanosecond intervals) to a date and time(128,129). The format of this field is not explained in the documentation for the SAV, and consequently implementation was challenging and time-consuming(105,107,130).

Dealing with the variable length records of the IndexMetricsOut.bin file was also challenging, as, although this was reasonably well explained in the documentation accompanying the SAV, the positions of the bytes encoding the lengths of the strings later on in the file were variable depending on the lengths of previous variable length strings(105). Therefore, a reasonable amount of initial parsing and calculation was required to calculate the length of each record in the bytearray, before parsing to extract required information could begin. A further difficulty arose because the struct library unpacks each character in a string as an individual entry, requiring the strings to be joined back together after unpacking. This necessitated identification of the indexes of the string entries in the final output from the parsers, to enable this subset of the final output to be retrospectively joined together into a single entry. Unfortunately, issues encountered during this process resulted in a solution that does not support records where these strings are empty (i.e. there is no sample name), and, as this situation does not currently arise so rectification is not considered urgent, this has been noted and scheduled for repair when time permits.

An overview of the functionality of the binary file parsers, illustrating the additional requirements of the ExtractionMetricsOut.bin and IndexMetricsOut.bin files is outlined in Figure 2.5.

Figure 4: An overview of the InterOp binary file parsers

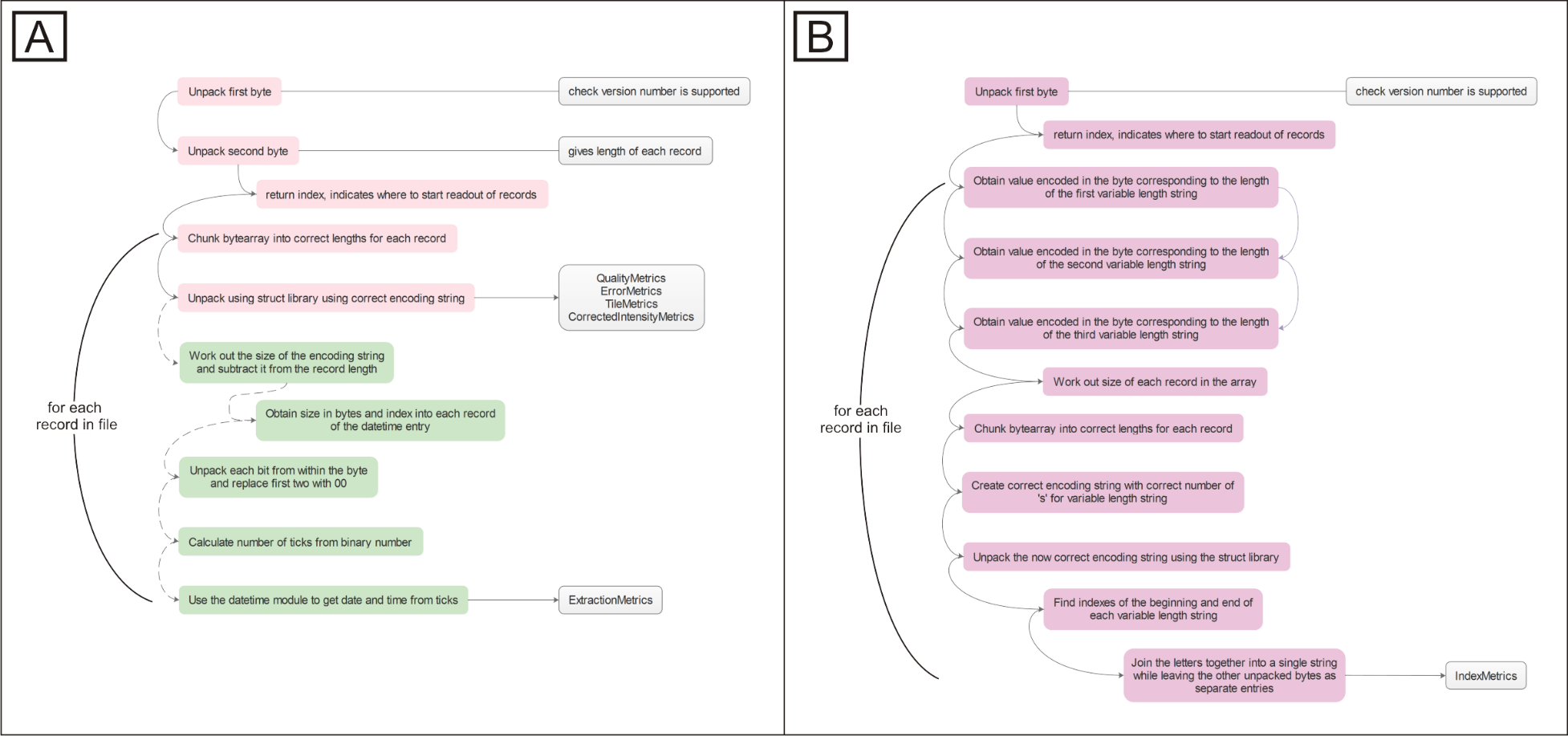


Figure 4: An overview of the functionality provided by the InterOp binary file parsers. Many of the files use the same parser with two files (ExtractionMetricsOut.bin and IndexMetricsOut.bin) requiring customised additional functionality. A) The functionality for the InterOp files; Quality Metrics, ErrorMetrics, TileMetrics, CorrectedIntensityMetrics in pink, with additional extra steps for ExtractionMetrics in green (dotted lines). The extra steps are required to unpack the date/time field which is encoded in the .NET format(130). B) The functionality provided for the IndexMetrics file (lilac). The record length changes for each sample, therefore there are a number of extra steps required. It is assumed that the index of the first value indicating the length of the first variable length string is at position 6. Purple arrows- the index of the subsequent value is dependent on having the previous value. The script currently does not handle cases where there is no sample name. It is necessary to join the letters of the sample name together as the struct library outputs each character as an individual letter(127).

The solution to the variable length of records in the IndexMetricsOut.bin file was not optimum as there are now a number of assumptions in the code, such as a sample must have a name and that the index of the byte encoding the length of the first string will always be at a certain hard-coded index, which means that the code is unlikely to be robust to a change in specifications. Fortunately, a check to determine whether the file version number is supported by the script was easy to implement (Figure 2.5, initial step), enabling unsupported file types to be easily identified and remedial action taken. Furthermore, the variable-length-record solution requires that the IndexMetricsOut binary files are iterated through a number of times, at first to extract information on length of certain variables (e.g. where there are variable sample names, the encoding string varies in length between samples), and subsequently using this information to correctly decode the file. This, in combination with the number of nested loops within the code, has serious implications for the run time and means that parsing a single file can take up to 300 seconds on a standard desktop machine (Intel Core2Duo P8600 2.4Ghz, 8GB RAM). There may be ways to reduce the number of times the IndexMetricsOut file is parsed and this will be explored when time permits. Additionally, it may be possible to remove some of the loops with alternative approaches to the problem, such as recursion. This could also help to reduce the order of the function and speed up the program execution(131,132).

Although some modularisation has been undertaken, the code is still not structured according to best object oriented programming principles, and therefore will contain inefficiencies. Improvements in code design could also increase the speed of file parsing(131,132). Furthermore, the parsers are currently lacking sanity-checking of values. As incorrect offsets in applying the struct library often results in data that is far outside the expected range of values or encodes a character that is clearly incorrect, this would be a relatively simple check to implement and would act as an immediate indicator of problems with parsing at runtime(127). This would also enable problems arising from particular datasets violating the assumptions of the code to be immediately identified and rectified. Changes to rectify these remaining design limitations within the code are planned for implementation as part of ongoing maintenance and upgrading(131,132).