

# The poRe GUIs

## Introduction

There are two poRe GUIs for working with Oxford Nanopore (ONT) MinION data:

- 1) poRe RT – for processing files in real-time as they are downloaded from the ONT cloud basecaller, Metrichor.
- 2) poRe Parallel – for fast data extraction from batches of existing data files, and plotting using extracted metadata files.

Both GUIs are written in Rstudio Shiny, and both use common functions for data extraction and plotting which are also usable outside the GUI framework (no shiny dependency).

Here, we first introduce features of the GUIs and their intended usage, before presenting a function reference of the nine R functions used across the GUIs.

## The poRe RT GUI

When poRe RT launches, it **immediately requests the user to select first a source folder** (which will normally be the Metrichor download folder), and **then a target folder** (into which processed files will be moved, in data-dependent subfolders).

**NOTE that on some operating systems, these will appear in the background. You may need to minimise your window to see them!**

Folder selection here uses the `dlgDir` cross-platform dialog box from the `svDialogs` package. poRe RT expects fast5 data files within the source folder to lie within either pass and fail folders, or barcoded folders within pass and fail folders (like the Metrichor download folder). The selected folders are displayed in text boxes in the GUI, as shown below. If a mistake has been made at this point, simply close and restart the GUI.

### PoRe RT GUI

Source Folder (Metrichor download folder)

C:/ZZZ Metrichor Test/fake\_metrichor\_download

☒ Run File Processing

Target Folder

C:/ZZZ Metrichor Test/pore\_rt download

☒ Show Plots

Real-time processing of files is initiated by selecting the [Run File Processing] tick box. Deselection halts processing but it can be continued by reselection. When real-time processing is running, the `process.fast5` function runs every two seconds and a data table of results is updated. As described in the function reference below, `process.fast5` processes small batches of fast5 files ( $\leq 200$  in poRe RT),

moving them to a GUI-specified location and using `extract.fast5` to get data from individual files. The five data elements required for plotting functionality are stored in a data table (`data.table` package).

Graphical output is controlled via the [Show Plots] tick box. When selected, plots are produced at the bottom of the GUI using the `meta.plot` function (see the function reference for details of the plots produced). The plots are updated every time `process.fast5` updates the central data table with metadata for plotting. This is typically every two seconds. An example of the GUI with plots is depicted below. This is running on some R9 Ecoli K12 data from Nick Loman's lab. The plots reflect the new rule for 2D protocol R9 data, where any file with a complement basecall goes into the pass folder.

## PoRe RT GUI

Source Folder (Metrichor download folder)

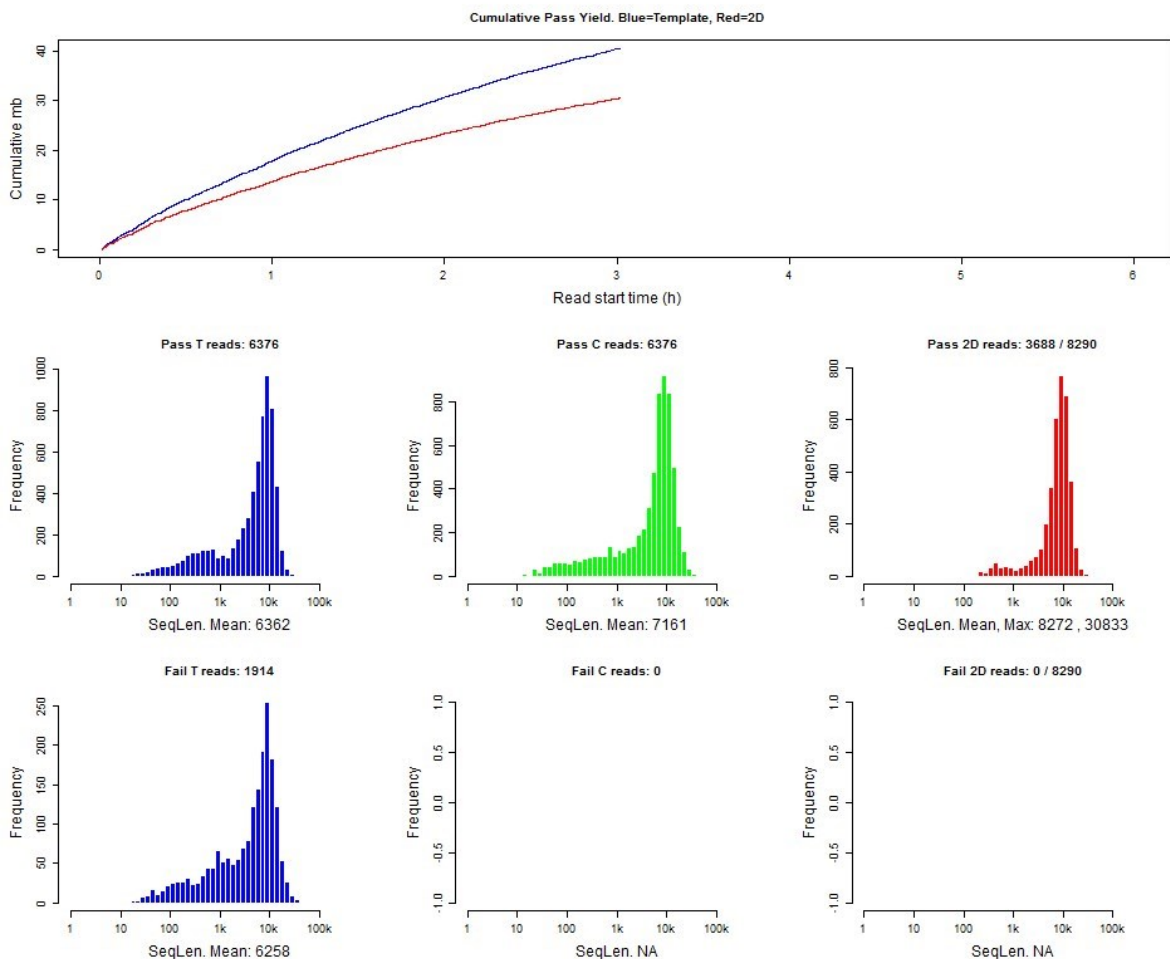
C:/ZZZ\_Metrichor\_Test/fake\_metrichor\_download

Target Folder

C:/ZZZ\_Metrichor\_Test/pore\_rt\_download

☒ Run File Processing

☒ Show Plots



Example console output for normal iterations is shown below:

```
.....37 54 51 49 22 44 45 49 50 48 54 49 54 55 50 50 52 52 44 53 53 49 53 57 56 55 49 52 52 56
52 55 59 50 58 60 55 57 58 61 64 60 58 60 65 67 67 61 64 72 71 68 70 67 71 71 68 69 69 67 76 69 77
63 68 74 63 76 66 66 75 80 79 70 71 70 66 67 70 72 69 68 72 74 75 68 74 71 .....
```

Error conditions from process.fast5 are also output to console. e.g.:

Unreadable: sce\_bio\_c03807\_MB\_0400\_FAA38345\_20150826\_1\_5247\_1\_ch37\_file10\_strand.fast5

Not Basecalled: sce\_bio\_c03807\_0820124747.fast5

Files that appear unreadable on one iteration (due to incomplete download) may be readable at the next. Repeated unreadable messages may indicate file corruption.

## The poRe Parallel GUI

When the parallel GUI launches, the user is presented with the following interface:

### PoRe Parallel GUI

The screenshot shows the PoRe Parallel GUI interface. It has a clean, modern design with a light gray background. At the top, there are two main sections: 'Source Folder' and 'Target Folder'. Each section has a text input field and a 'Choose' button below it. Below these, there are three columns of settings. The first column, 'Select output file type(s)', has checkboxes for 'Fastq' (checked), 'Fasta', and 'Meta', with a 'Select All/None' button below. The second column, 'Select Dataset(s)', has checkboxes for '2D' (checked), 'Template', and 'Complement'. The third column, 'If output filenames match', has radio buttons for 'Rename' (selected) and 'Overwrite'. A 'Run Data Extraction' button is located to the right of these settings. Below the settings, there is a 'Status' section with a large text area. At the bottom, there is a 'Metadata File' section with a text input field, a 'Choose Metadata File' button, and a 'Show/Update Plots' button.

This GUI is designed for two separate modes of operation: data extraction and plotting.

### Data Extraction

To run data extraction, the first step is to choose source and target folders by clicking on the respective buttons.

**NOTE that on some operating systems, these will appear in the background. You may need to minimise your window to see them!**

As for poRe RT, folder selection uses the `dlgDir` cross-platform dialog box from the `svDialogs` package. Unlike poRe RT, this program only processes files directly in the selected source folder (rather than pass/fail subfolders).

Next, the types of output file, and the specific datasets to extract should be selected, and the selection check boxes lie directly below the Source Folder selection area of the GUI. With the default

options, only the 2D fastq datasets will be extracted to one or more bulk fastq files. Additional file type options enable creation of fasta and/or metadata files and the choices are independent (e.g. metadata only can be output). For fastq or fasta outputs, the Dataset selection boxes determine which fastq/fasta datasets to extract and write to file. There will be up to 6 files here (3 datasets x 2 file types) for each run; output files are divided into run name subfolders.

The [Select All/None] button selects or deselects all file types and datasets for convenience.

To the right of the select dataset boxes, radio buttons enable differing behaviour in the case that there are files in the target locations which match the names of output files due to be produced by the program (.fasta, .fastq, or meta.text files only). By default, existing files are renamed by appending .old on the file name. If the Overwrite option is selected, existing files are simply overwritten.

Finally, the [Run Data Extraction] button launches the data extraction process according to the selected options. Data extraction here proceeds in batch-parallel mode. First, files are divided by run name, with each run processed separately. For each run, batches of files of size 5040 (or less) are processed in parallel using `extract.fast5` and `parLapply` from the `parallel` package. This batch parallel processing ensures that memory requirements are limited, and progress can be tracked on large datasets. After data is extracted from each batch of files, the batch data is written out to file(s), using `write.table` (meta) and `write.seq` (fastq and fasta).

The figure below shows the GUI in the middle of a batch data extraction of 8323 R9 fast5 files. A progress bar section at the top gives both a text and graphical indication of progress through runs and batches. Progress updates are also output to the console.

The screenshot displays the PoRe Parallel GUI interface. At the top right, a status bar indicates "Processing Run 1 of 1 Batch 2 of 2". The main area is divided into several sections:

- Source Folder:** A text box containing "C:/ZZZ\_Metrichor\_Test/temp\_source4" and a "Choose Source Folder" button.
- Target Folder:** A text box containing "C:/ZZZ\_Metrichor\_Test/temp" and a "Choose Target Folder" button.
- Select output file type(s):** Three checked checkboxes for "Fastq", "Fasta", and "Meta", along with a "Select All/None" button.
- Select Dataset(s):** Three checked checkboxes for "2D", "Template", and "Complement".
- If output filenames match:** Two radio buttons, with "Rename" selected and "Overwrite" unselected.
- Run Data Extraction:** A button to initiate the process.
- Status:** A text box at the bottom showing "Updated Source Folder. Found 8323 fast5 files".

The Status text box is used to give important feedback to the user, but it cannot update during a batch parallel run, so it greys out. If [Run Data Extraction] is clicked before selecting both source and target folder, status shows:

#### Status

You need to select both source and target folders for Data extraction. Please try again.

If folders have been selected but no output file type is selected, [Run Data Extraction] gives:

#### Status

You need to select at least one file type. Please try again.

If fastq or fasta file type is selected, but no dataset is selected, [Run Data Extraction] gives:

#### Status

You need to select at least one dataset to extract fastq or fasta data. Please try again.

Finally, completion of data extraction (for all file types and datasets) gives:

#### Status

Done. Processed 8323 fast5 files in 61.9s

Metadata extraction takes longer than fastq/fastq extraction. For comparison, on the same 8323 R9 files, 2D fastq extraction alone takes around 15 s using 4 cores on a hyperthreading i7-4600U laptop CPU at 2.1 GHz, with the files on a local SSD (1/4 of the time for complete extraction).

### Plotting

The plotting functionality in the poRe parallel GUI is independent from data extraction. All it requires is an existing metadata file (tab separated text file), which can be created by poRe RT or poRe parallel. A metadata file is selected by clicking [Choose Metadata File], using `dlgDir` as elsewhere. The selected file is shown in the accompanying text box.

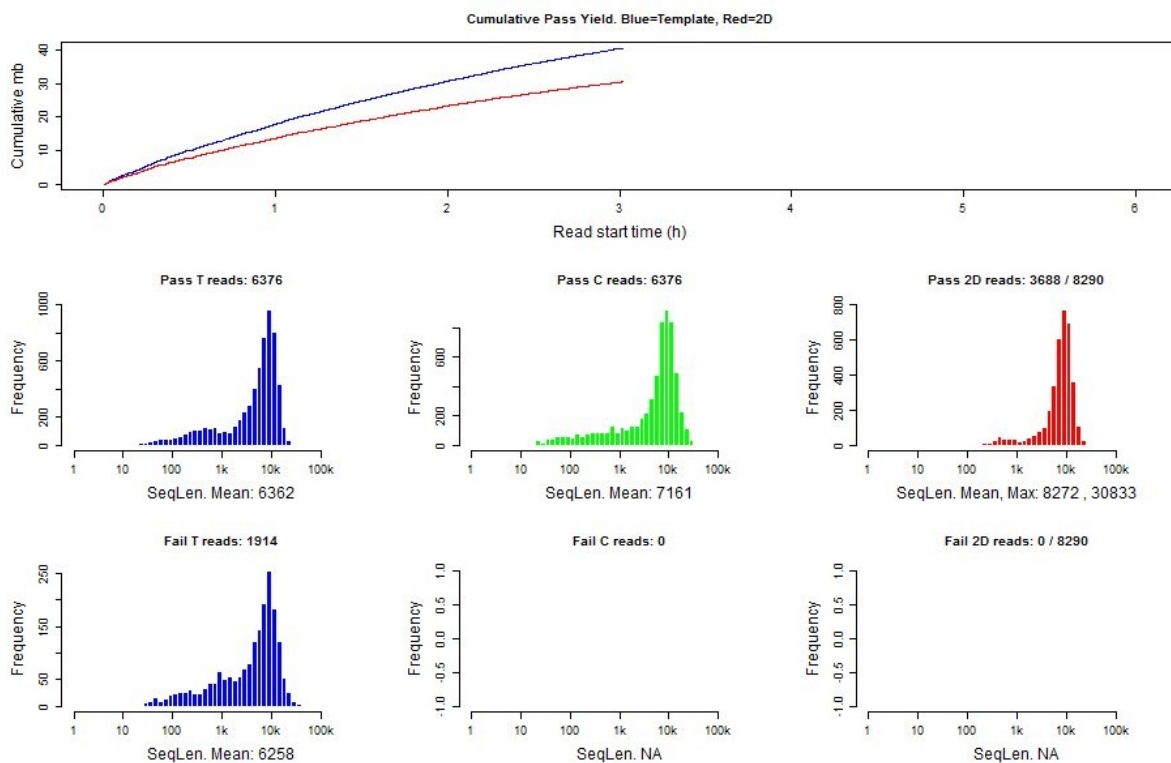
Once a metadata file is selected, clicking [Show/Update Plots] reads data from the file using `read.delim` and then runs `meta.plot` on the data frame obtained, with output shown at the bottom of the GUI. Example output is shown below.

## Metadata File

nanopore2\_R9\_MinKNOW\_0.51\_Ecoli\_K12\_MG1655\_lambda\_2258\_meta.txt

Choose Metadata File

Show/Update Plots



Finally, clicking [Show/Update Plots] before selecting a metadata file shows an error status message:

## Status

You need to select a metadata file for plotting. Please try again.

## Function Reference

F5dir

### Description

Uses file.path and dir.create to construct a file path and recursively create the corresponding directory.

### Usage

```
F5dir(path, name)
```

### Arguments

path	Character vector containing a single path name.
name	Character vector containing a folder name.

### Details

Uses file.path to first construct a well-formatted file path, in a platform-independent way, from an existing path and a folder name. Then uses dir.create to create a directory at the specified location, with recursive=TRUE and showWarnings=FALSE.

### Value

Same return value as file.path with the same inputs

rhdf5 convenience wrappers

### **Description**

These functions provide a simple syntax for reading attributes and datasets from an hdf5 file.

### **Usage**

```
F5Aread(f5obj, name)
```

```
F5Aread_by_name(f5obj, objname = ".", name)
```

```
F5Dread(f5loc, name)
```

### **Arguments**

- |                      |  |
|----------------------|--|
| <code>f5obj</code>   | An object of class <code>H5IdComponent</code> representing an H5 object identifier (file, group or dataset). See rhdf5 documentation for detail. |
| <code>f5loc</code>   | An object of class <code>H5IdComponent</code> representing an H5 location identifier (file or group).  |
| <code>name</code>    | The name of the attribute or dataset (character).  |
| <code>objname</code> | The name of the object an attribute belongs to.  |

### **Details**

These functions are convenience wrappers for the rhdf5 low-level interface to the HDF5 C-library. They each perform an open, a read and a close in sequence. `F5Aread` uses `H5Aopen`, `H5Aread` and `H5Aclose`. `F5Aread_by_name` uses `H5Aopen_by_name` but is otherwise identical. `F5Dread` uses `H5Dopen`, `H5Dread` and `H5Dclose`.

### **Value**

`F5Aread` and `F5Aread_by_name` return an array with attribute data. `F5Dread` returns an array with dataset data.



meta.plot

## Description

Produces a compound plot from a fast5 metadata table, with a cumulative pass yield graph for 1D template and 2D data, and sequence length histograms for both pass and fail data. **Usage**

meta.plot(DT)

## Arguments

DT	A data frame or data table with 5 required named columns: DT\$lenT (int, 1D template sequence length), DT\$lenC (int, 1D complement sequence length), DT\$len2D (int, 2D sequence length), DT\$pass (logical, true only for files passing all quality filters), DT\$start_time (int, start time in seconds).
----	--

## Details

The compound plot is arranged into 2-3 rows. The top row is always a cumulative yield plot, which uses DT\$lenT, DT\$len2D, DT\$pass and DT\$start\_times to show the cumulative yield expressed as total (template and 2D) pass data megabases (mb) over read start time (after the run start time) in hours. Rows 2-3 show sequence length histograms. If pass data are present, then row 2 shows pass histograms, while row 3 shows fail histograms if fail data is also present. If only fail data is present, the pass data row of plots will be absent and row 2 shows fail data results. From left to right, there are 3 plot columns on the pass and fail rows, showing sequence length histograms for template, complement and 2D basecalls, respectively. All sequence lengths are plotted on a standard log scale, from 1 to 100k (this may change to range up to 1 million with nanopore advances). Both cumulative yield and histogram plots are colour-coded with blue=template, green=complement and red=2D.

## Value

No return value

extract.fast5

## Description

Extracts sequence data and metadata from a fast5 file.

**Usage** extract.fast5(filename,flags)

## Arguments

filename      Character vector filename with full path

flags          Named logical vector with 6 members specifying extraction options: flags\$fastq (get fastq data), flags\$fasta (get fasta data), flags\$meta (get metadata), flags\$dt (get template data), flags\$dc (get complement data), flags\$d2d (get 2D data).

## Details

Initial conditionals check that the file is a readable hdf5 file and that it has basecalled data (sequences). Failure on either of these tests returns an error string (error = "unreadable"; error = "no\_basecall"). If all three basic checks pass, then data is extracted from the file according to the flag values specified. For example, template fastq data is extracted only when flags\$dt and flags\$fastq are both TRUE. Where fasta data is requested, then fastq strings are converted into nanopolishcompatible fasta strings (with the full file path in the header) using the get.fasta function.

## Value

Named character vector with 18 members (12 metadata and 6 sequence data). Names and default values as follows:

```
data_line <- c(filename='fast5',chan_num='-1',read_num='-1',start_time='-1',pass='F',lenT='0',
lenC='0',len2D='0',run_id='no_run_id',read_id='no_read_id',barcode='no_bc',exp_start_time='-1',
fastqT="",fastaT="",fastqC="",fastaC="",fastq2D="",fasta2D="")
```

get.fasta

### Description

Small, utility function to convert a fastq string into a nanopolish-compatible fasta string with the full file path in the header. **Usage** get.fasta(fastq\_splits,filename)

### Arguments

fastq_splits	Character vector containing the four lines of a fastq string as separate elements.
filename	Character string with full path filename, for fasta header

### Details

First, the location of the space is detected in the first element of fastq\_splits (i.e. the header). This space divides the read name and the filename as returned by metrichor (with no file path information). The full path filename provided as an argument is then substituted into the header and the initial '@' character is swapped for '>'. Finally, the header is concatenated with the second element of fastq\_splits, which contains the base sequence.

### Value

Character string

process.fast5

## Description

Used in the RT GUI only, this function processes small batches of fast5 files, moving them to a GUIspecified location and using extract.fast5 to get data from individual files. The five data elements required for meta.plot are stored in a data table (data.table package). **Usage**

```
process.fast5(DT,max_files=200,flags=c(fastq=TRUE,fasta=TRUE,meta=TRUE,dt=FALSE,dc=FALSE,d2d=TRUE))
```

## Arguments

DT	Data Table with 5 columns required for meta.plot (see that function for detail)
max_files	Maximum number of files to process before returning an answer (allowing plotting)
flags	Logical vector as required for extract.fast5 (see that function for detail)

## Details

This is a shiny-specific function, and all of the body is isolated to avoid unwanted reactivity. In addition to conventional input arguments, it works with several shiny reactiveValues: RV\$f5\_list (character vector as returned by list.files), RV\$nf5 (int, length of RV\$f5\_list), RV\$np (int, index of current file for processing), and RV\$npoc (int, total number of files processed). These variables are held as reactiveValues so that they persist across multiple calls to this function.

If there are no fast5 files (in RV\$f5\_list) yet to be processed (RV\$np>RV\$nf5), a check for any new files is run and the reactiveValues are updated accordingly. If there are no fast5 files in the userdefined source folder, (RV\$nf5 == 0), then a dot is output to the console. Otherwise, a batch of up to max\_files size of fast5 files is processed. Files are first moved to a run name folder in the userdefined target folder (run names are extracted from the file names), and then data are extracted using extract.fast5. Duplicate files are moved to a separate 'duplicates' folder, while files with no basecalled data are moved to a 'no\_basecall' folder. When files have been processed, the number of files processed is output to the console for a visual update on progress.

## Value

Data Table in the same format as the input argument DT

write.seq

### Description

Used in the parallel static GUI only, this function writes sequence data (fastq/fastq) to file in batches.

**Usage** write.seq(x,file,append=FALSE) **Arguments** x Character vector of sequences

file Character string containing the name of the file to write data to append

Logical determining whether data should be appended (passed to cat) **Description**

If append is TRUE, x is appended to existing sequence data in the target file. If append is FALSE, the target file exists and rename is selected in the GUI, the existing file is renamed. Finally, if append is false, x is written to the target file without appending.

### Value

No return value