Parsing Bruker-Ultraflex TOF Mass Spectrometry Data Using the William and Mary Bruker Parser Package

By William Cooke, Maureen Tracy and Dariya Malyarenko
The College of William and Mary

1 Overview of the William and Mary Parser for Bruker-Ultraflex TOF Mass Spectrometry Data 1
1.1 Introduction 1
1.2 Background 1
1.3 Methods 2
1.4 Examples 6
1.4.1 Example 1: Parse data from a single run 6
1.4.2 Example 2: Parse and concatenate data from multiple runs 7
1.5 Concerns 9
1.6 Conclusions and Recommendations 10
2 Routines in the W&M Bruker Ultraflex Parser Package 11
2.1 BrukerParser 11
2.2 ParseAndSave 12
3 Data Stuctures Generated 13
tofList 13
tofListMetaData 14
4 Acknowlegement 17
5 References 17

1 Overview of the William and Mary Parser for Bruker-Ultraflex TOF Mass Spectrometry Data

1.1 Introduction

Bruker Daltonics Ultraflex MALDI Mass Spectrometers save data in binary format in a unique directory structure. (For simplicity, this data will be referred to herein as either Bruker or Bruker-Ultraflex data). In order to facilitate analysis of Bruker data from large studies (e.g. involving numerous patient samples) in R, it is necessary to parse the Bruker data and associated experimental parameter files into R objects. The purpose of this package is to provide this capability and to save the R objects for further signal processing and statistical analysis.

1.2 Background

Protein profiling using Time-Of-Flight (TOF) Mass spectrometry (MS) is an important technique in proteomics, as it can direct the identification of biologically significant

species (biomarkers). Bruker Daltonics Ultraflex MALDI Mass Spectrometers provide fast scanning over broad mass ranges which is useful for the analysis of multiple biological samples. While peak broadening and low signal-to-noise are issues for high mass ranges, signal processing tools (background subtraction, integrative down-sampling, deconvolution filtering, pedestal removal, peak detection and alignment) have been developed [1, 2] to address these challenges. The availability of broad-mass data allows the application of new signal processing techniques such as the detection of ionization satellites (distinct from molecular ions) and reconstruction of molecular protein signatures that further improves selectivity and sensitivity for molecular ions [3, 4].

A typical large-scale study employing MALDI-TOF includes sample purification, using different affinity capture agents (e.g., C3 or IMAC magnetic beads [3]) and spotting several replicates of the purified protein mixture with matrix on a MALDI plate (384 spots, numbered (1-24) x (A-P)) for profiling and identification. The chemical preparation steps are preferably done with a programmable robotic bioprocessor for better control and reproducibility. A single clinProt (Bruker) bioprocessor run allows spotting of 96 samples (limited by number of bioprocessor wells) in 4 replicates (row or column pattern). As an example, a clinical study for 200 samples with three replicates would require at least 7 bioprocessor runs. The MALDI spectra may then be obtained for different mass ranges (e.g., 1-20, 15-100, and 20 -150 kDa) to ensure optimal data acquisition through balancing resolution and sensitivity [3]. The maximum allowed buffer size for single Bruker Ultraflex spectrum is 0.5 Mb. Therefore, for a typical experiment (including pooled QC samples), as in our example above, one may expect a data set size of 0.3-0.5 Gb per mass range.

1.3 Methods

Our package contains two routines. The core routine, “BrukerParser,” reads binary TOF data (obtained in either Linear or Reflector mode) and associated experimental parameter files, and parses data and meta-data into R structures, tofList and tofListMetaData. This involves accessing files in different locations within a data directory tree and reading different file formats. The second routine, “ParseAndSave” calls the “BrukerParser” routine as directed by the parameter list provided in an “OptionsAndParameters.txt” file and saves the tofList and tofListMetaData structures that are returned in .Rdat files. Parameters in “OptionsAndParameters” can be edited to select options for parsing data from multiple bioprocessor runs, concatenating parsed data and printing memory usage information. While parsing the data from a study, it may be useful to reflect experimental design, e.g., by separately saving the data from different affinity purification agents and in different mass ranges.

Figure 1 illustrates the Bruker data directory structure for Linear TOF data and highlights the locations of the binary data file and associated experimental parameter files that are read by the “BrukerParser” routine. The upper portion of Figure 1 shows the data directory (in this example, ~C3ValidationExtractSmall/RobotRun1/2-100kDa) which contains subdirectories with spot names (“0_A1” and “0_B1”), a calibration directory,

Figure 1: Directory tree containing Bruker Daltonics Ultraflex MALDI Mass Spectrometer output files for Linear TOF data. In each image, a directory is highlighted on the left and its contents are displayed on the right. The upper image highlights the data directory which contains subdirectories with spot names, a calibration directory, and optional experimental parameter files ("sample.xml", and file with extensions ".axe", ".par", and ".isset"). The lower image highlights the directory two levels down in the spot subdirectories (~"spot name"/1/1SLin/fid) that contains the "fid" binary data file, and the required "acqu" acquisition parameter file. For Reflector TOF data, the "fid" file would reside in a "1SRef" directory rather than a 1SLin directory.

and optional experimental parameter files ("sample.xml", and files with extensions ".axe", ".par", and ".isset"). The "sample.xml" file is generated by clinProt robot, and provides the mapping between the sample well on the bioprocessor and the spot position on the MALDI plate. Additional information can be provided by the operator in this file, including sample ID, clinical data (e.g., disease group), affinity surface used for purification, and replicate number. Other instrumental XML files are initially located in several different directories of flexControl software and reflect the parameters that are set in different panes of the instrument controlling software: e.g., ".par" records mass range, source voltage and gain settings specified in "Detection" pane; ".axe" specifies information related to laser settings and automated spot reading protocol in the "Auto-execution" pane; and ".isset" contains detector gain and laser power settings of the "Setup" pane. The user has to "save" the corresponding parameter files after making

changes, in order to preserve the current settings, and be able to load them later for reproducible analysis [3]. To parse the above instrumental setting information, the corresponding parameter files have to be copied to the experimental directory manually from the flexControl method directories. Since they are not saved automatically, the parser code has been designed to tolerate their absence (making them optional). When they are missing, the corresponding attribute structures will not be populated, and the meta-data fields will be set to N/A. The user has the option (discussed below) of updating these meta-data fields for use in further analysis. For instance, when the "sample.xml" file is missing, the user may want to manually assign any relevant sample ID or disease group information for the study.

The binary mass spectrum data files (required) are named "fid" and are located two levels down (in /1/1Lin/ for Linear data or /1/1SRef/ for Reflector data) in the spot subdirectories along with another (required) instrumental parameter file "acqu". The lower image in Figure 1 displays the location of the binary data file for spot "0_A1". (~C3ValidationExtractSmall/RobotRun1/2-100kDa/0_A1/1/1SLin/fid). There is also a "fid" file in a parallel location in the "0_B1" subdirectory (~C3ValidationExtractSmall/RobotRun1/2-100kDa/0_B1/1/1SLin/fid) that is not shown. The "acqu" file contains much of the summary for experimental acquisition parameters. While the "acqu" file contains parameters in common with the "optional" files discussed above, it is less comprehensive, especially for detector and laser pattern settings. Both the "fid" and the "acqu" files are automatically saved by flexControl in each spot/sample directory. A third file, also saved automatically by flexControl, is called "proc" and is saved two directories below the "fid" and "acqu" files. The "proc" file contains the processing parameters for flexAnalysis software, which can be ignored, if custom processing is planned. The only meta-data currently parsed from the proc file is the calibration date. The examples detailed in Section 1.4 below, provide the parameter assignments and R commands required to run the parser for the directory structure illustrated in Figure 1.

Figure 2 shows the output of the parser for the data directory "~C3ValidationExtractSmall/RobotRun1/2-100kDa". The location of the "tofListRun1.Rdat" and "tofListMetaDataRun1.Rdat" files in the data directory is shown in the upper image in Figure 2. (The string "Run#" has been added to distinguish output from different bioprocessor runs if multiple runs are parsed.) The lower portion of Figure 2 provides images of partial listings of the tofList and tofListMetaData R structures. For this example, the tofList contains two spectrum vectors (for two spots, "0_A1" and "0_B1"). In the tofListMetaData structure, rows correspond to the spectra and columns correspond to meta-data attributes (See Section 3, "Data Structures Generated" for the listing of the 61 attributeNames and descriptions.) Each screen shot shows two conventions for accessing data in these structures. TOF spectrum vectors can be accessed in the tofList by spectrumName or by index, and metaData values can be accessed in the tofListMetaData by spectrumName and attributeName or by row and column indices. Accessing data by spectrumNames and attributeNames is preferred as it is possible for one structure to be indexed differently from the other. (This could occur if spectrum

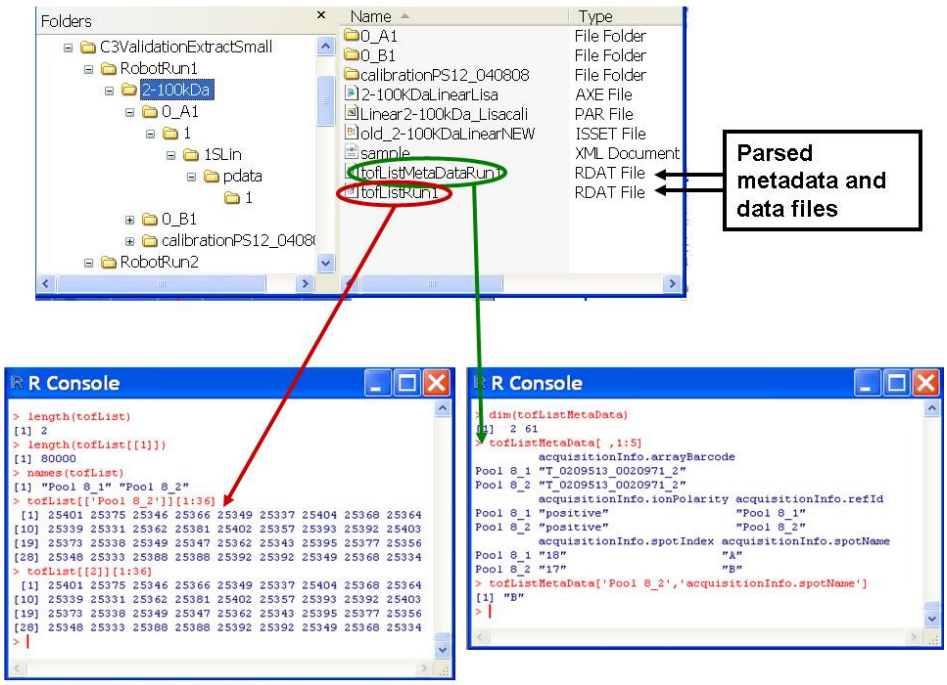

Figure 2: Output files generated by BrukerParser for data directory "~C3ValidationExtractSmall/RobotRun1/2-100kDa". The upper image shows the location of "tofListRun1.Rdat" and "tofListMetaDataRun1.Rdat" in the data directory. The lower images show partial listings of the tofList and tofListMetaData structures. Two addressing conventions are shown, by indices and by names. The tofList is a list of time-of-flight mass spectrum vectors addressable by spectrumName (preferred) or index. The tofListMetaData is a data.frame with rows and columns addressable by spectrumName and attributeName (preferred) or indices.

vectors are sorted according to a clinical parameter and the meta-data is not during processing.).

Figures 1 and 2 illustrate input and output for parsing data from a single data directory When parsing data from multiple runs, tofList and tofListMetaData ".Rdat" files for each run are saved in the each of the corresponding data directories. If data is concatenated (an option to be used with caution as explained in the "Concerns" section below), the concatenated tofList and tofListMetaData files are saved in the data directory for the final run parsed. Figure 3 illustrates the output found in the second data directory when data from two runs are parsed and concatenated. (The first data directory would contain the same files as illustrated in Figure 2.) The optional concatenated tofList is 4 vectors long since each run contains data for two spots. Similarly, the optional concatenated tofListMetaData would have 4 rows and 61 columns. (The number of meta-data attributes is unchanged.)

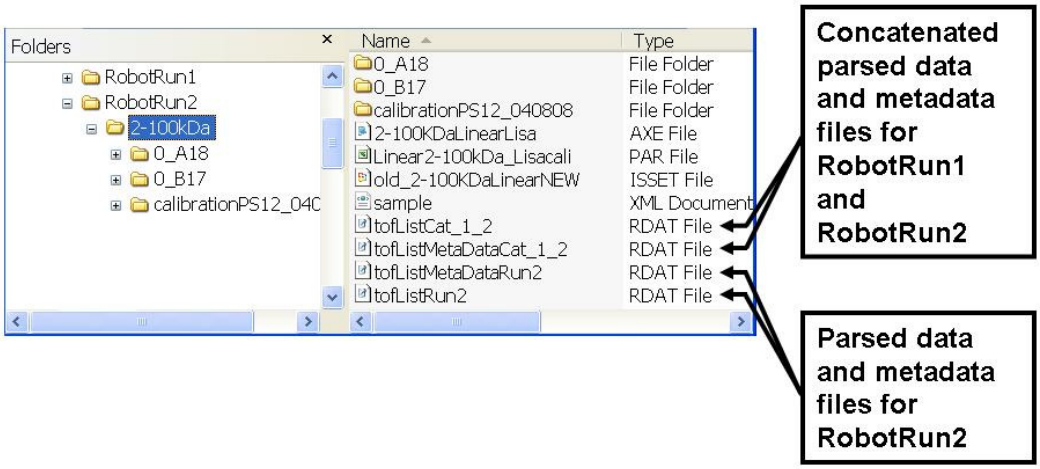

Figure 3: Output in the second data directory when data from two runs are parsed and concatenated. When data from multiple runs are parsed, tofList and tofListMetaData files for individual runs are saved in their respective data directories and concatenated files (if opted for) are saved in the data directory from which the final data was parsed.

1.4 Examples

Two examples are included in this package. The subdirectory "Examples" contains a Bruker output directory tree and the two OptionsAndParameters text files. These latter files contain parameters required by the routines in the package and are intended to be edited by the user for their purposes. "C3ValidationExtractSmall" contains data for two runs for two spots each (as seen in Figures 1-3) and will be used for both examples.

1.4.1 Example 1: Parse data from a single run

The parameters contained in "OptionsAndParametersParse1Run.txt" are shown below.

----------OptionsAndParametersParse1Run.txt----------

```
# Specify laboratory where data was obtained
ParserParams$dataSource <- "EVMS"

# Specify if multiple runs are to be parsed, "yes" or "no"
ParserParams$multipleRuns <- "no"

# Specify whether or not to concatenate tofLists, "yes" or "no"
ParserParams$concatLists <- "no"

# Specify the indices of the runs to parse.
# MUST APPEAR IN DATA DIRECTORY PATH
ParserParams$runIndices <- 1

# Specify data directory path to the left of the index (or complete path if
```

```
# parsing one run)

directory = system.file("Examples", package = "WMBrukerParser")
ParserParams$dataDirLeft <- paste(directory,"/C3ValidationExtractSmall/RobotRun1/2-100kDa "
",sep="")

# Specify data directory path to the right of the index (if there is one).
ParserParams$dataDirRight <- ""

# Specify whether or not memory and time usage are to printed during parsing
ParserParams$printMemoryUse <- "no"

----------End OptionsAndParametersParse1Run.txt----------
```

To execute the Example 1, type:

```
> directory = system.file("Examples", package = "WMBrukerParser")
> source(paste(directory,"/OptionsAndParametersParse1Run.txt",sep=""))
> ParseAndSave(ParserParams)
```

Parsed "tofListRun1.Rdat" and "tofListMetaDataRun1.Rdat" files should be available in "~C3ValidationExtractSmall/RobotRun1/2-100kDa" as shown in Figure 2.

1.4.2 Example 2: Parse and concatenate data from multiple runs

Parameters in "OptionsAndParametersParseAndCat2Runs.txt" that differ from those in "OptionsAndParametersParse1Run.txt" used in Example 1 follow:

```
----------------------------------------------------------------------------------------------

# Specify if multiple runs are to be parsed, "yes" or "no"
ParserParams$multipleRuns <- "yes"

# Specify whether or not to concatenate tofLists, "yes" or "no"
ParserParams$concatLists <- "yes"

# Specify the indices of the runs to parse.
# MUST APPEAR IN DATA DIRECTORY PATH
ParserParams$runIndices <- c(1,2)

# Specify data directory path to the left of the index (or complete path if
# parsing one run)
directory = system.file("Examples", package = "WMBrukerParser")
ParserParams$dataDirLeft <- paste(directory,"/C3ValidationExtractSmall/RobotRun",sep="")

# Specify data directory path to the right of the index (if there is one).
ParserParams$dataDirRight <- "/2-100kDa"

# Specify whether or not memory and time usage are to printed during parsing
ParserParams$printMemoryUse <- "yes"
```

NOTE: The “run branch” is not required to be one directory up from the data directory. The use of two parameters (ParserParams$dataDirLeft and ParserParams$dataDirRight) to construct the data directory path allows the "run branch" to be located at the level of the data directory or anywhere above.

To execute the Example 2 type:

```
> directory = system.file("Examples", package = "WMBrukerParser")
> source(paste(directory,"/OptionsAndParametersParseAndCat2RunsRun.txt",sep=""))
> ParseAndSave(ParserParams)
```

Parsed “tofListRun1.Rdat” and “tofListMetaDataRun1.Rdat” files should be available in "~C3ValidationExtractSmall/RobotRun1/2-100kDa” as shown in Figure 2. Parsed “tofListRun2.Rdat”, “tofListMetaDataRun2.Rdat”, “tofListCat_1_2.Rdat”, and “tofListMetaDataCat_1_2.Rdat” files should be available in "~C3ValidationExtractSmall/RobotRun2/2-100kDa” as shown in Figure 3.

Here two runs have been concatenated. Additional runs can be easily included in the concatenation by adding indices to the ParserParams$runIndices parameter in the "OptionsAndParameters.txt" file. (The indices need not be in order but must appear in the path to the data directory.) For N runs:

```
ParserParams$runIndices <- c(runindex1, runindex2, runindex3,…runindexN);
```

Since the option to print the memory use during parsing has been selected (ParserParams$printMemoryUse <- "yes") for this example, memory (in MB) and time information are printed after loading or removing data for each run. The screen shot on the left of Figure 4 shows the R console when executing Example 2. For comparison, the screen shot on the right of Figure 4 shows the R console during execution of the same data without concatenation. The difference in memory used after the lines “tofListMetaData for Run2 has been removed” in the two screen shots in Figure 4, 1.259 MB, should be approximately equal to the memory used for the concatenated tofList and tofListMetaData.

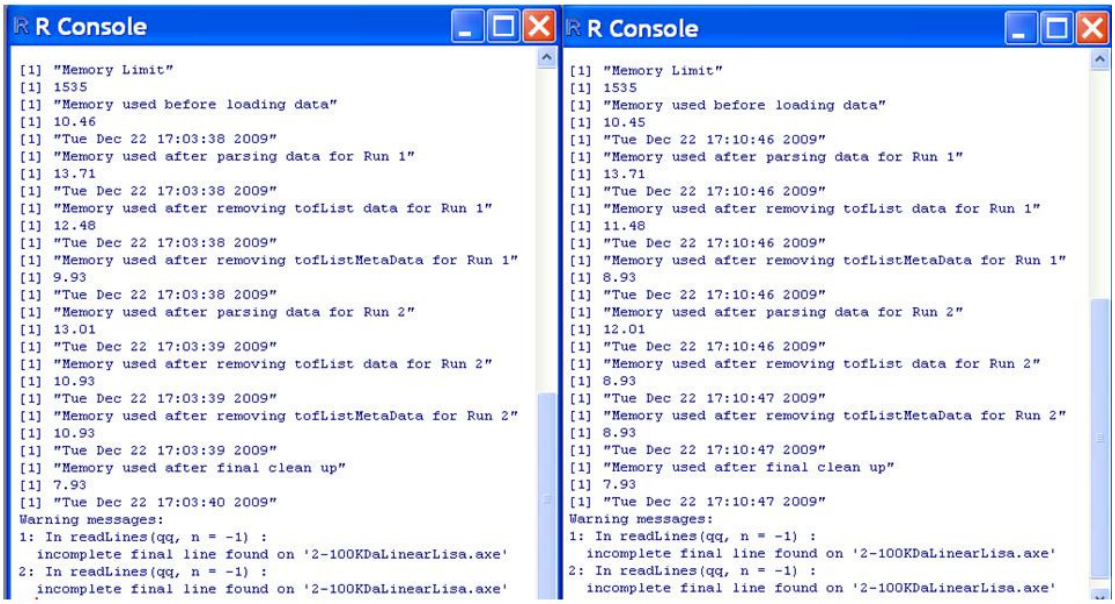

Figure 4. R Console during parsing of data from two runs with option to print memory use selected. The screen shot on the left is for the execution of Example 2. The screen shot on the right is for parsing the same data as in Example 2 but without concatenation. The differences between memory used at each step reflects the increasing size of the concatenated tofList and tofListMetaData objects.

1.5 Concerns

Since MS signal processing includes an alignment step, it would be ideal to parse all spectra from all runs in a given study into a single tofList. However the data files are large and there is a limit to the amount of memory that R has available (particularly during further processing which requires twice the memory as required to simply load the data). According to the R Documentation (on Package *utils* version 2.8.0) for memory.size and memory.limit, the Windows version of R is usually limited to 2GB of memory. To insure the efficient use of memory, the ParseAndSave routine calls the BrukerParser for one run at a time and removes data once it has been parsed, saved and (if opted for) concatenated. Even with this precaution, it is possible to exceed the memory available during parsing. As discussed above, an option has been included to print memory use during parsing to monitor the growing memory requirements of the concatenated tofList and tofListMetaData objects.

At this point, it is helpful to consider the size of the original Bruker data files, the size of the parsed files, and the memory required by R to load the parsed files for Example 2 above.

The size of the Bruker data files on disk are:

~C3ValidationExtractSmall/RobotRun1/2-100kDa/0_A1/1/1SLin/fid: 0.467 MB
~C3ValidationExtractSmall/RobotRun1/2-100kDa/0_B1/1/1SLin/fid: 0.467 MB
~C3ValidationExtractSmall/RobotRun2/2-100kDa/0_A18/1/1SLin/fid: 0.467 MB
~C3ValidationExtractSmall/RobotRun2/2-100kDa/0_A17/1/1SLin/fid: 0.467 MB

The size of the resulting parsed data and meta-data files on disk are:

tofListRun1.Rdat: 0.388 MB
tofListRun2.Rdat: 0.395 MB
tofListCat_1_2.Rdat: 0.782 MB

Memory used to load the parsed data and meta-data files into R (found using command memory.size(max=FALSE) before and after loading each file) :

tofListRun1.Rdat: 1.00 MB
tofListRun2.Rdat: 1.00 MB
tofListCat_1_2.Rdat: 2.00 MB

While these values may differ for other computers, it appears that the parsed data files take up less disk space than the raw data files. However, the memory required to load parsed data files into R is greater by more than a factor of two times their size on disk. The concatenated tofList files are approximately as large as the sum of the individual files on disk and in R. The difference in maximum memory used for parsing with and without concatenation, 10.93 MB and 8.93 MB, shown in the screenshots in Figure 4, corresponds to the size of the concatenated tofList, 2.00 MB. (The metaData structures in this example require less than 0.00 MB memory.)

1.6 Conclusions and Recommendations

This package provides the utility for parsing Bruker data into R structures on a run by run basis. Prior to use of the option to concatenate data from multiple runs, the size of the resultant concatenated file should be estimated (for an upper limit, multiply the number of spectra by the size in MB of the "fid" files) and compared with half of the available memory in R (allowing for the doubled memory requirements for processing). If the estimate of the size of the concatenated file approaches half of the memory available, it would be wise to run "ParseAndSave" with ParserParams$multipleRuns <- "yes" , ParserParams$concatLists <- "no," and ParserParams$printMemoryUse <- "yes". The resultant listing of memory usage in R, considered with the observed size of parsed files on disk will aid in deciding which runs to include in concatenation. Note that there is the option of including down-sampling in the subsequent signal processing steps [1] which would effectively compresses the data for each run separately, and allow concatenation of the full data set after down-sampling before alignment.

2 Routines in the W&M Bruker Ultraflex Parser Package

2.1 BrukerParser

Description

This routine reads the binary “fid” Bruker files, along with other associated experimental files and creates R structures tofList and tofListMetaData.

Usage

BrukerParser(dataSource, dataDirectory)

Arguments

dataSource - Name of data acquisition site, for example, “EVMS”
dataDirectory - Directory containing spot subdirectories

Details

This routine expects the following contents in the dataDirectory:
spot subdirectories with names like "0_A1" and "0_L5", each containing:
EITHER (for Linear TOF data):
/1/1SLin/fid - the binary data file (required)
/1/1SLin/acqu - the acquisition information file (required)
/1/1SLin/pdata/1/proc - the processing file (required)
OR (for Reflectron TOF data):
/1/1SRef/fid - the binary data file (required)
/1/1SRef/acqu - the acquisition information file (required)
/1/1SRef/pdata/1/proc - the processing file (required)
a subdirectory with “calibration” in the name (optional)
a file with the extension “.par” (optional)
a file with the extension “.axe” (optional)
a file with the extension “.isset” (optional)
a file named “sample.xml” (optional)

If any of the optional files are missing, the user has the option updating meta-data fields in the parsed meta-data structure, tofListMetaData. If the sample.xml file is missing, a field used in naming data and meta-data structures is missing and placeholders will be used. In this case, the user may wish to update structure names. Examples of the procedures to update data field and names are given below Section 3: Data Structures Generated.

If the final line of the “.axe” file is not blank, R will generate a warning message such as:
“Warning message:
In readLines(qq, n = -1) : incomplete final line found on '0-20KDaLin030308.axe' “

This has no effect on the reader and can be ignored.

Value

tofList - A list of time-of-flight mass spectrum vectors addressable by spectrumName.
tofListMetaData - A data.frame containing string values for experimental meta-data, with rows and columns addressable by spectrumName and attributeName.

Author

William Cooke, College of William and Mary

Note

Called by ParseAndSave

2.2 ParseAndSave

Description

This routine calls the BrukerParser to parse data from individual runs. It has options for parsing data from multiple runs and for concatenating the parsed data structures. Individual and concatenated tofList and tofListMetaData files are saved as .Rdat files.

Usage

ParseAndSave(ParserParams)

Arguments

ParserParms – List containing:
- ParserParams$dataSource - laboratory where data was obtained
- ParserParams$multipleRuns - number of runs to parse
- ParserParams$ concatLists - whether or not to concatenate tofLists (“yes” or “no”)
- ParserParams$ runIndices - indices of the runs to parse (must appear somewhere in the data directory path)
- ParserParams$dataDirLeft - Portion of the data directory path to the left of the run index (or full path name if a single run is being parsed)
- ParserParams$dataDirRight - Portion of the data directory path to the right of the run index (if it exits)
- ParserParams$ printMemoryUse - if "yes" memory usage and date stamps throughout parsing will be printed

Details

Since the data files are large, it may not be desirable or possible to concatenate data parsed from all runs in an experiment. The option to print memory usage through the parsing of multiple runs allows the user to consider which files to select for concatenation. Prior to selecting the option to concatenate data, it would be wise to parse all the data of interest, examine the memory usage throughout parsing in R and size of the resulting .Rdat files on the disk, keeping in mind the doubling of memory required for the further processing of this data.

"tofListRun#.Rdat" and "tofListMetaDataRun#.Rdat" files are saved in "dataDirectory" (the directory containing the spot subdirectories) for "Run#". Concatenated files are saved in the directory of the final data parsed. For instance if runs 1, 3, and 2 were selected for concatenation (ParserParams$runIndices<-c(1,3,2)), the files "tofListCat_1_3_2.Rdat" and "tofListMetaDataCat_1_3_2.Rdat" would be found in the dataDirectory for the Run 2.

After tofList and tofListMetaData structures (individual and concatenated) are saved, they are removed from the workspace.

Value

tofList - A list of time-of-flight mass spectrum vectors addressable by spectrumName (saved as .Rdat file)

tofListMetaData - A data.frame containing string values for experimental meta-data, with rows and columns addressable by spectrumName and attributeName. (saved as an .Rdat file)

Author

Maureen Tracy, College of William and Mary

Note

Calls BrukerParser

3 Data Stuctures Generated

tofList

tofList - A list of time-of-flight mass spectrum vectors addressable by spectrumName.

To access a spectrum vector:

```
spectrum <- tofList[[ spectrumName ]];
```

For example:

```
spectrum <- tofList[[ "pool 2_8" ]];
```

Alternately, spectrum vectors can be addressed by spectrum index. This option should be used with caution since data can be reordered during processing.

To update a spectrum's vector:

```
tofList[ spectrumName ] <- list(spectrum);
```

To update the tofList names (desirable if default assignments were used due to missing sampleInfo.sampleNames meta-data):

```
newSpecNames<-list();
numSpec<-length(tofList);
for (i in 1:numSpec) { newSpecNames[i]<-paste("spec", i , sep="") };
names(tofList)<-newSpecNames;
```

A partial listing of a sample tofList structure is shown in the lower left of Figure 2.

tofListMetaData

A data.frame containing string values for experimental meta-data, with rows and columns addressable by spectrumName and attributeName.

To access a value:

```
sampleName <- tofListMetaData[ spectrumName, attributeName ];
```

For example:

```
sampleName <- tofListMetaData["pool 2_8", " sampleInfo.sampleName " ];
```

Alternately, meta-data can be addressed using spectrum and attribute indices. Again, this option should be used with caution since data can be reordered during processing.

To update a value (which is desirable if tofListMetaData includes NAs due to missing optional parameter files):

```
tofListMetaData[ spectrumName, attributeName ] <- "Test";
```

To update the tofListMetaData rownames (desirable if default assignments were used due to missing sampleInfo.sampleNames)

```
newSpecNames<-list();
numSpec<-length(tofList);
for (i in 1:numSpec) { newSpecNames[i]<-paste("spec", i , sep="") };
row.names(tofListMetaData)<-newSpecNames;
```

A partial listing of a sample tofListMetaData structure is shown in the lower right of Figure 2.

Meta-data attributes include:

acquisitionInfo.arrayBarcode – target (Bruker plate) identifier string, includes target and target serial number - *Required*
acquisitionInfo.ionPolarity – "positive" or "negative" - *Required*
acquisitionInfo.refId – spectrum identifier, (same as experiment.id), used for spectrum vector names in tofList and rows names in tofListMetaData (generated by concatenating strings: sampleInfo.sampleName,"_", and replicateNumber. If sampleInfo.SampleName is unavailable, a default assignment of the directory name is used.) - *Optional*
acquisitionInfo.sourceFile – file name (with path) containing binary mass spectrum data, (from directory structure) - *Automatic*
acquisitionInfo.spotIndex – spot index on MALDI plate - *Automatic*
acquisitionInfo.spotName – spot name on MALDI plate - *Automatic*
adcGain.high – digitizer gain factor - *Optional*
adcGain.low – linear detector voltage - *Optional*
array.affinityType – capture agent used for purification of biofluid samples - *Optional*
deflector.mode – 1=linear, 2=quadratic - *Optional*
experiment.id – same as acquisitionInfo.refId - *Optional*
instrument.instrumentType – "Ultraflex 3" (hard coded) - *Automatic*
instrument.instrumentVendor – name of instrument vendor - *Required*
instrument.serial – instrument serial number - *Required*
instrumentSpecificSettings.adcBandwidth – digitizer bandwith limit - *Optional*
instrumentSpecificSettings.adcOffset – digitizer offset, depends on spectrum type and gain factor - *Optional*
instrumentSpecificSettings.adcScale – digitizer sensitivity, depends on gain factor - *Optional*
instrumentSpecificSettings.detectorBaseVoltage – detector base voltage, depends on spectrum type - *Optional*
instrumentSpecificSettings.laserIntensityBaseRange – minimum laser power - *Optional*
instrumentSpecificSettings.laserShots – number of laser shots per spot - *Required*
instrumentSpecificSettings.timeZero – intial time in counts (calculated) - *Required*
instrumentSpecificSettings.TOFmode – spectrometer mode: LINEAR or REFLTOR – *Optional*
instrumentSpecificSettings.warmingShots – number of warming laser shots - *Optional*
laserIntensity.units – % (hard coded) - *Automatic*
laserIntensity.value – maximum laser power, percent of full scale - *Optional*
laserIntensityWarming.units – % (hard coded) - *Automatic*
laserIntensityWarming.value – warming Laser power, percent of full scale - *Optional*
mass.units – "Da" (hard coded) - *Automatic*
mass.value – low mass cut off for matrix suppression - *Optional*
massCalibration.calibrationSpectrumFile – calibration folder name. (from folder name

containing “calibration.” If unavailable "?" is assigned as default) - *Optional*

massCalibration.dateCalibrated – calibration date - *Required*

massCalibration.equation – "c1*((T0+(X-1)*Tdelta)/U)^2+c0*((T0+(X-1)*Tdelta)/U)+c2" (hard coded) - *Automatic*

massEnd.units – mass units, “Da” (hard coded) - *Automatic*

massEnd.value – highest mass in acquisition range – *Optional*

massStart.units – mass units, “Da” (hard coded) – *Automatic*

massStart.value – lowest mass in acquisition range - *Optional*

param.c0 – linear coefficient in calibration equation - *Required*

param.c1 – quadratic coefficient in calibration equation - *Required*

param.c2 – constant term in calibration equation - *Required*

param.Mode – calibration mode: 1=linear, 2= quadratic - *Optional*

param.T0 – digitizer delay in ns - *Required*

param.TDelta – dwell time in ns - *Required*

param.U – nanoseconds/milliseconds conversion factor, "1e6" (hard coded) - *Automatic*

replicateNumber – index of occurrence of sampleInfo.sampleName - *Automatic*

sampleInfo.groupName – clinical group that sample belongs to, used for classification analysis - *Optional*

sampleInfo.sampleDescription – additional sample description - *Optional*

sampleInfo.sampleName – sample identifier (read from sample.xml, If sample.xml, or the field is missing, the directory name is used as the default value) - *Optional*

sampleInfo.sampleSource – laboratory where data was acquired (user supplied input parameter: ParserParams.dataSource) - *Required*

software.version – version of XACQ software - *Required*

spottingInfo.spotProtocol – laser movement patern during data acquisition - *Optional*

timeOfFlightData.dateCreated – data file creation date (from date stamp on fid file) - *Automatic*

timeOfFlightData.domain – data domain, "time" (hard coded) - *Automatic*

timeOfFlightData.encoding – data encoding, "base64" (hard coded) - *Automatic*

timeOfFlightData.end – number of data points in spectra - *Required*

timeOfFlightData.offset – initial value for offset of spectra, will be updated during alignment procedure, "0.0" (hard coded) - *Automatic*

timeOfFlightData.pairOrder – Order of measurement pairs for TOF data, e.g., time – intensity, "t-int" (hard coded) - *Automatic*

timeOfFlightData.scale – initial value for linear scale coefficient for spectra, will be updated during alignment procedure, hard coded: "1.0" (hard coded) - *Automatic*

timeOfFlightData.start – index for onset of data acquisition, hard coded: "1" – *Automatic*

4 Acknowlegement

This research was supported by NIH National Cancer Institute R01 Grant CA126118 from the Advanced Proteomics Platforms and Computational Sciences Program within the Clinical Proteomics Initiative of the National Cancer Institute (PI: Malyarenko).

5 References

[1] Malyarenko, D. I., et al., *Rapid Commun. Mass Spectrom* (2006) **20**, 1670–1678

[2] Tracy, M.B., et al., *Proteomics* (2008) **6**, 4517-4524

[3] Gatlin-Bunai, C. L., et al., *J Proteome Res* (2007) **6**, 4517-4524

[4] Malyarenko, D.I., et al., *Rapid Commun. Mass Spectrom* (2009) **23**, in press