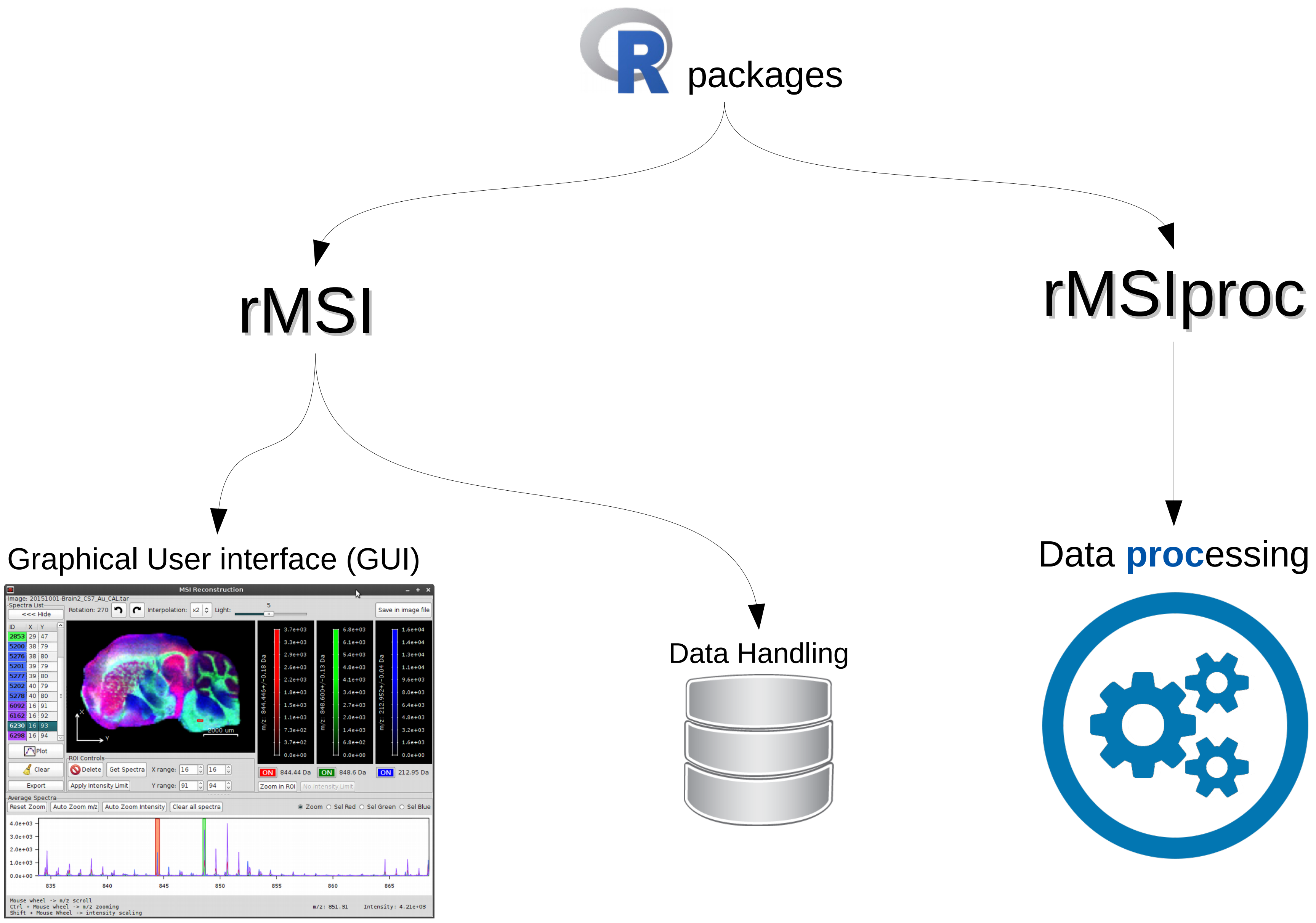
rMSI & rMSIproc Tutorial

Pere Ràfols Soler

Introduction



R basics

- Since rMSI and rMSIproc are R package it is recommended to have some R language background.
- Both packages provides graphical user interfaces (GUI) so many tasks can be achieved without a deep knowledge of R.
- A short R introduction is provided in this tutorial to get startted using rMSI and rMSIproc.
- More in depth information about R is available online:

  https://cran.r-project.org/doc/manuals/r-release/R-intro.html

  https://www.rstudio.com/online-learning/#R

R basics

- **Comments:** Anything starting with the character # is a comment in R.
  - Comments are not executed by R interpreted.
  - Comments are useful to explain what a part of code is doing.

R code example: Comments

```
# This sentence is a comment because it starts with the # character.
# This means that I can write whatever I want inside R language if I start
# using the # character. Note that I need start each commented line with #.
# R will simply ignore any line starting with #.

However, if I write a comment sentence without the #, R will raise an error
because it is not able to understand it.
```

R basics

- **Variable:** A place to store data (a number, a character, a string, a matrix, a list, an object… anything)
  - A variable is named with a meaningful string (e.g. MyVar)
  - A variable is assigned using the <- operator

R code example: Variables

```
MyVar <- 23 #My variable now contains the number 23
print(MyVar) #This prints the content of MyVar in R console

MyVar <- “this is a string var” #My variable now contains a text string
print(MyVar) #Now it will display: “this is a string var” instead of 23
```

R basics

- **Vector:** An R data type which contains a collection of ordered items with the same data type.
  - Valid data types are: numbers, characters and strings.
  - A vector is stored using a variable
  - Elements are acceded by its position inside a vector (index)

R code example: Vectors

```
MyVec <- 1:10 #Create a vector that contains numbers 1 to 10
print(MyVec) #Prints the whole vector ( 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
print(MyVec[3]) #Prints only the third element of MyVec (3)

MyVec <- c(10, 33, 20) #Create a vector using the concatenate operator
print(MyVec[2:3]) #Print MyVec elements 2 to 3, so 33, 20 is printed

MyVecStr <- c(”ab”, “cde”, “fghi”) #Create a vector of strings
```

R basics

- **Matrix:** A collection of numbers arranged in a matrix form.
  - A matrix is stored in a variable.
  - Matrix elements are accessed by its row and column position.
  - A matrix is created using the matrix() function.

R code example: Matrix

```
MyMat <- matrix(1:6, nrow = 2, ncol = 3) #Create a matrix of  2 rows and 6
# columns containing numbers from 1 to 6

print(MyMat)
# This is the contents of the matrix
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

MyMat[2,1] <- 99 # Change the value of the item located at row 2 column 1

print(MyMat)
# This is the contents of the matrix
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]   99    4    6
```

R basics

- **List:** A collection of arbitrary elements.
  - A list is stored in a variable.
  - The [[ ]] operator is used to access elements by index.
  - The $ operator is used to access elements by name.

R code example: List

```r
MyList <- list() #Create an empty list
MyList <- list( a = 32, b = “abc” ) #Create a list with some named elements

MyVar <- MyList[[1]] #Get the first element and store it in a variable
MyVar <- MyList$a #We can also access the same element by its name

# We can use a list to store other list. This is useful to create a tree.
# In the next example, MyListTop contains two lists.
MyListTop <- list()
MyListSub1 <- list( name = “Element1”, value = 1 )
MyListSub2 <- list( name = “Element2”, value = 2 )
MyListTop[[1]] <- MyListSub1
MyListTop[[2]] <- MyListSub2

# We can access the second list “Element2” value doing the following:
MyListTop[[2]]$value
```

R basics

- **Function:** A chunk of code packed in a way to make it callable any time without having to re-write it all again and again.
  - R packages are made of functions.
  - Data is delivered to a function using parameters.
  - Data is retrieved from a function using return values.

R code example: Function

```
# Declaration of a function that just sums to value.
# input data or parameters are a and b.
# the return value of this function is the sum of a and b.
MyFun <- function( a, b ) {
    return( a + b )
}

# Using a function for just summing to values is not very useful but
# now we can call MyFun function each time we want to sum two values.

MyVar <- MyFun( 3, 2) # MyVar will contain the number 5
```

R basics

- **Script:** We can put all our commands inside an R script instead of writing them to the R console.
  - We can execute a part of a script or all of it.
  - An R program is a collection of R scripts.
  - RStudio is recommended for writing, executing and managing scripts.

Getting rMSI and rMSIproc

rMSI: https://github.com/prafols/rMSI/

rMSIproc: https://github.com/prafols/rMSIproc

Installation

First install rMSI package following the instructions in the github page.

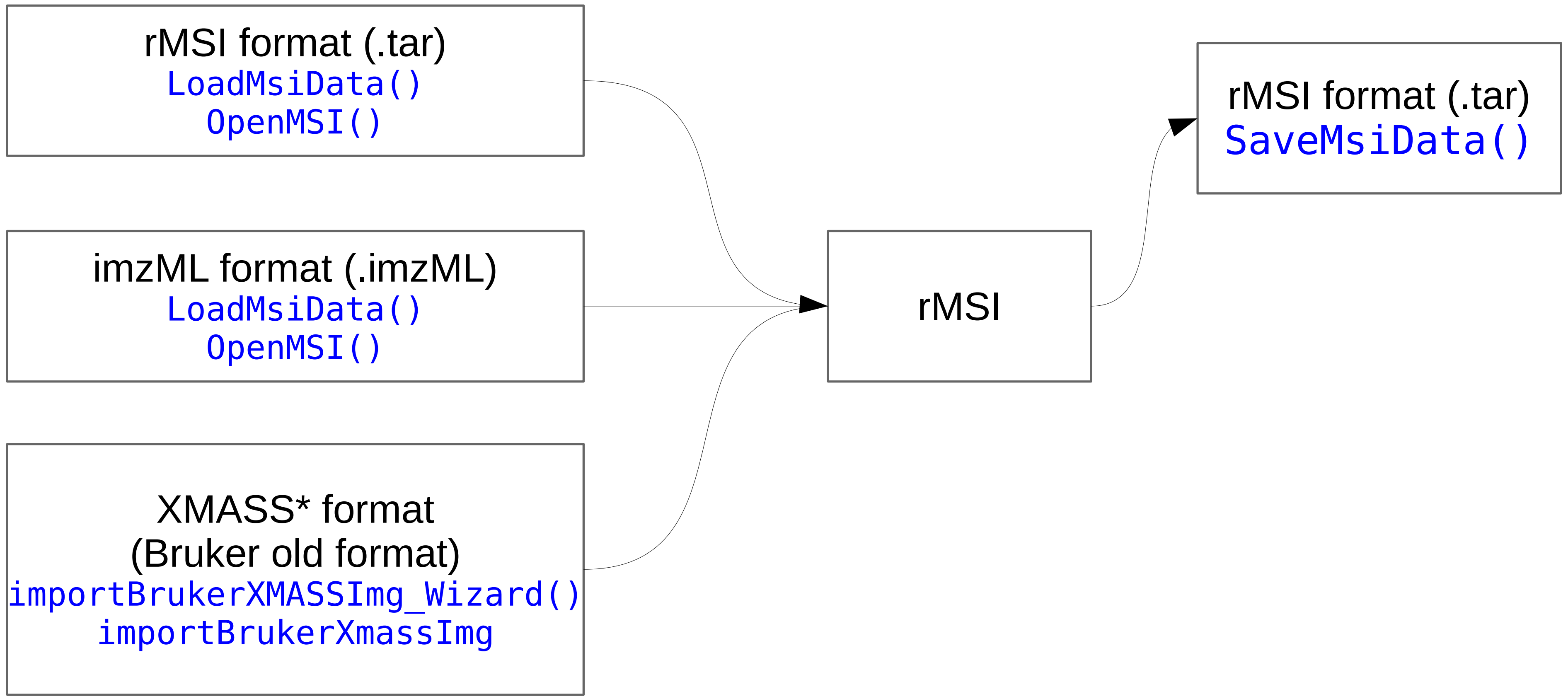
Then, use the devtools packages to install rMSIproc directly from github (fill the ref = x.x using the last released version):

```
devtools::install_github("prafols/rMSIproc", ref = "x.x")
```

rMSI: Data input/output

MSI data reading

MSI data writing



*XMASS is a deprecated format. Import functions using XMASS are implemented to provide backward compatibility with older versions of FlexImaging which are not able to export in imzML. Avoid it when possible and use imzML instead.

Loading MSI data with rMSI

R code example: Loading MSI data

```
# Load a MSI dataset using the GUI
MyDataList <- rMSI::OpenMSI()

# This will open a graphical form where up to two MSI datasets can be loaded
# We can access each one of the up to two datasets using the $ operator
MyData <- MyDataList$img1

# MyData is a variable that contains the first loaded dataset

# Load a MSI dataset in rMSI format by command line
MyData <- rMSI::LoadMsiData("/path/to/my/Msimage.tar")

# We can also load a MSI dataset in imzML forat using the same function
MyData <- rMSI::LoadMsiData("/path/to/my/Msimage.imzML")

# MyData is a variable that contains an rMSI object with the loaded dataset
```

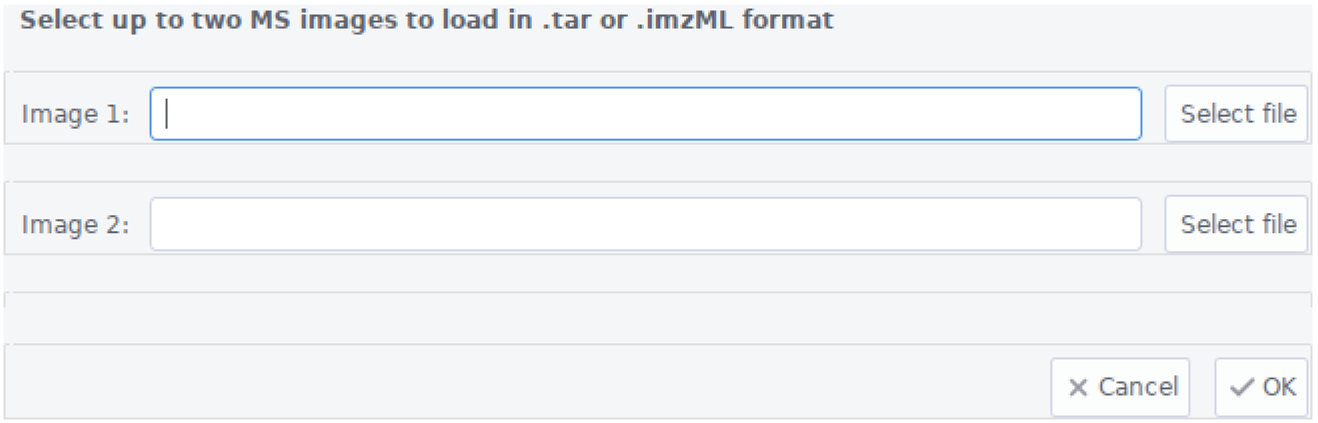

Select up to two MS images to load in .tar or .imzML format

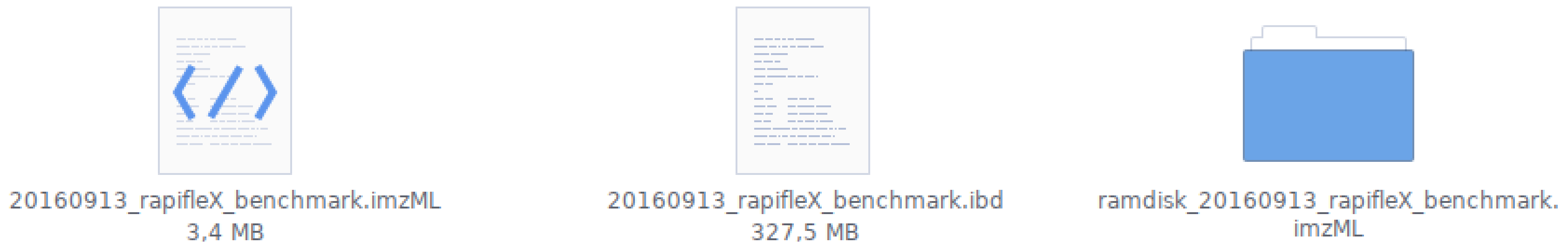
Image 1:

Image 2:

OpenMSI() GUI

rMSI ramdisk

- When a MSI dataset is loaded it is not loaded in the computers main memory (RAM).
- MSI data can be much more large than RAM, so a the hard disk drive (HDD) is used instead.
- When a imzML file is loaded, it is converted to rMSI format an stored inside a folder in the HDD.
- When an rMSI .tar file is loaded, it is unpacked inside a folder in the HDD.
- Imported XMASS files are converted to rMSI format an stored inside a folder in the HDD.
- This folder in the HDD is called the **ramdisk**.
- The **ramdisk** is used by rMSI to access MS data efficiently even is data is larger than RAM.
- The **ramdisk** can be safely deleted if we need to retrieve some HDD space.


20160913_rapifleX_benchmark.imzML
3,4 MB
20160913_rapifleX_benchmark.ibd
327,5 MB
ramdisk_20160913_rapifleX_benchmark.imzML

The MSI dataset in imzML files

The rMSI ramdisk

Exploring MSI data with rMSI

R code example: Exploring MSI data

```
# User-friendly way: Open rMSI main GUI
# OpenMSI() function will automatically launch the GUI when data is loaded
MyDataList <- rMSI::OpenMSI()

# We can load the main GUI from console anytime using MSIWindow() function
rMSI::MSIWindow(MyDataList$img1)

# R-way: Using R scripts, commands and so on…
# For example, to plot a ion distribution using the standard R plot:
rMSI::plotMassImageByPeak(MyDataList$img1, mass.peak = 845, tolerance = 0.1)

# For advanced usage, we need to know a little bit more about rMSI internals
```

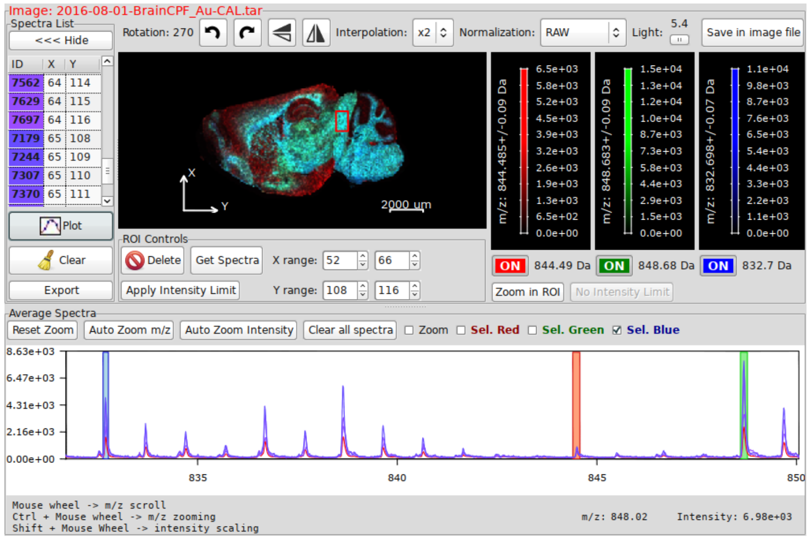


Image: 2016-08-01-BrainCPF_Au-CAL.tar

Rotation: 270 Interpolation: x2 Normalization: RAW Light: 5.4

ID	X	Y
7562	64	114
7629	64	115
7697	64	116
7179	65	108
7244	65	109
7307	65	110
7370	65	111

X range: 52 66
Y range: 108 116

m/z: 844.485+/-0.09 Da: 6.5e+03, 5.8e+03, 5.2e+03, 4.5e+03, 3.9e+03, 3.2e+03, 2.6e+03, 1.9e+03, 1.3e+03, 6.5e+02, 0.0e+00 — ON 844.49 Da

m/z: 848.683+/-0.09 Da: 1.5e+04, 1.3e+04, 1.2e+04, 1.0e+04, 8.7e+03, 7.3e+03, 5.8e+03, 4.4e+03, 2.9e+03, 1.5e+03, 0.0e+00 — ON 848.68 Da

m/z: 832.698+/-0.07 Da: 1.1e+04, 9.8e+03, 8.7e+03, 7.6e+03, 6.5e+03, 5.4e+03, 4.4e+03, 3.3e+03, 2.2e+03, 1.1e+03, 0.0e+00 — ON 832.7 Da

8.63e+03, 6.47e+03, 4.31e+03, 2.16e+03, 0.00e+00

```
Mouse wheel -> m/z scroll
Ctrl + Mouse wheel -> m/z zooming
Shift + Mouse Wheel -> intensity scaling
m/z: 848.02    Intensity: 6.98e+03
```

rMSI main GUI

rMSI object internals

A MSI dataset is loaded in an rMSI using an R list object with the following elements:

- **$name:** The MSI data name, generally the filename.
- **$uuid:** A unique identifier to avoid mixing data.
- **$mass:** A vector with the mass axis. All pixels share the same mass axis.
- **$size:** A vector of two elements with the number of pixels in X and Y directions.
- **$pos:** A matrix of two columns with the spatial XY coordinates of all pixels.
- **$posMotors:** The same as $pos but with original instrument coordinates.
- **$pixel_size_um:** The pixel resolution in microns.
- **$mean:** The average spectrum of the whole dataset
- **$data:** A structure pointing to spectral data stored in rMSI ramdisk.

  DO NO TRY TO ACCESS $data DIRECTLY UNLESS YOU KNOW rMSI FORMAT REALLY WELL.

- **$normalizations (optional):** A list of intensity normalization coefficients. If this available the main GUI will provide a drop-down menu to select the normalization.

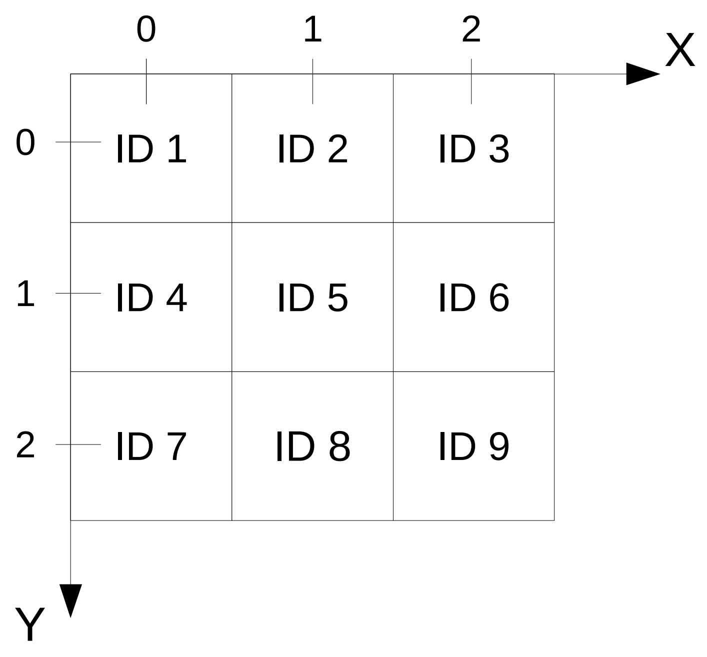
rMSI object internals: $pos matrix

The position matrix ($pos) is an important part of the rMSI data format.

- It keeps track of where each spectrum/pixel is spatially located in a MSI dataset.
- It keeps track of pixels by assigning a unique ID to each one.
- It keeps track of data by associating each ramdisk chunk with its corresponding pixel.

<u>Example:</u> A 3 by 3 pixels MS image

The MS image pixel positions



The pos matrix for the 3x3 dataset

X	Y	ID
0	0	1
1	0	2
2	0	3
0	1	4
1	1	5
2	1	6
0	2	7
1	2	8
2	2	9

rMSI data access

R code example: data access

```
# MS data can be directly accessed using loadImgChunkFromIds() and
# loadImgChunkFromCoords() functions to load a bunch of selected pixel spectra to
# an R matrix. This functions load data into standard R variables so, be careful to
# not load to many data in your computer’s memory. For example, lets suppose I want
# to load some spectra of pixels with ID‘s 34, 56 and 96:
MySpectra <- rMSI::loadImgChunkFromIds(MyData, c(34, 56, 96))

# If I don't know the ID's but I know the XY coordinates the spectra can be also
# access expressing the XY coordinates as complex numbers, for example:
MySpectra <- rMSI::loadImgChunkFromCoords(MyData, complex(real = c(10, 12, 34),
              imaginary = c(5, 9, 12)))

# In both cases, an R matrix is returned. Each row of the matrix corresponds to one
# spectrum using the same ordering as the ID’s or XY coords were provided.
# So you can get your first spectrum intensities by doing:
MyFirstIntensities <- MySpectra[1, ]

# The mass axis is common to the whole image and is available at the mass field:
# MyData$mass. This spectrum can be plotted using the rMSI spectra viewer GUI:
rMSI::plotSpectra(mass = myData$mass, intensity = MyFirstIntensities, col = "red")
```

rMSIproc

- rMSI is used to read/write MSI data
- Spectra smoothing using Savitzky-Golay
- Label-free alignment using cross-correlation
- Mass re-calibration
- Intensity normalization
- Peak-picking
- ROI export
- Data merging

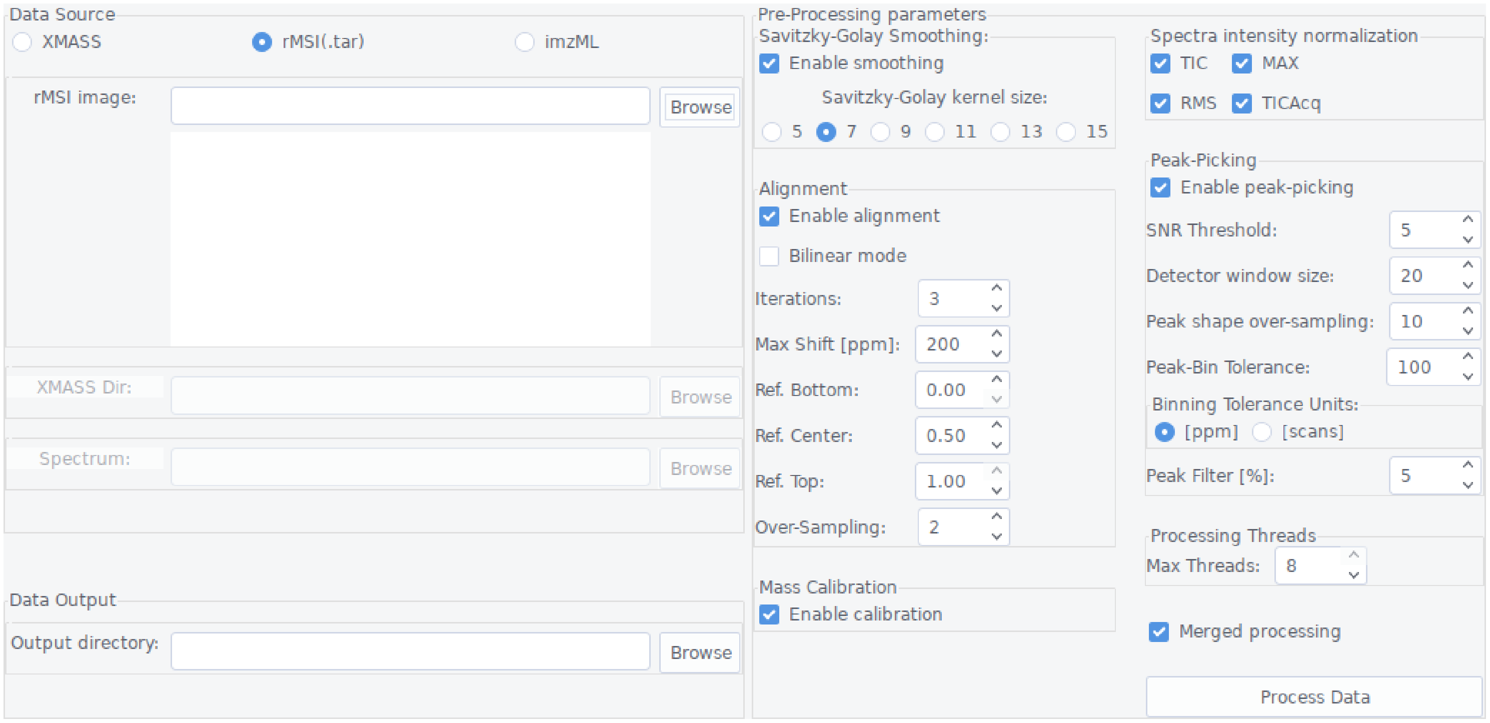
rMSIproc

Almost all rMSIproc functions are available trough a GUI.

So it is recommended to use the GUI instead of writing R scripts.

Just call the following command to launch the GUI:

```
rMSIproc::ProcessWizard()
```



Data Source
XMASS [✓ rMSI(.tar)] imzML

rMSI image: Browse

XMASS Dir: Browse

Spectrum: Browse

Data Output
Output directory: Browse

Pre-Processing parameters

Savitzky-Golay Smoothing:
✓ Enable smoothing
Savitzky-Golay kernel size:
5 [✓ 7] 9 11 13 15

Alignment
✓ Enable alignment
Bilinear mode

Iterations:	3
Max Shift [ppm]:	200
Ref. Bottom:	0.00
Ref. Center:	0.50
Ref. Top:	1.00
Over-Sampling:	2

Mass Calibration
✓ Enable calibration

Spectra intensity normalization
✓ TIC ✓ MAX
✓ RMS ✓ TICAcq

Peak-Picking
✓ Enable peak-picking

SNR Threshold:	5
Detector window size:	20
Peak shape over-sampling:	10
Peak-Bin Tolerance:	100

Binning Tolerance Units:
[✓ [ppm]] [scans]

Peak Filter [%]: 5

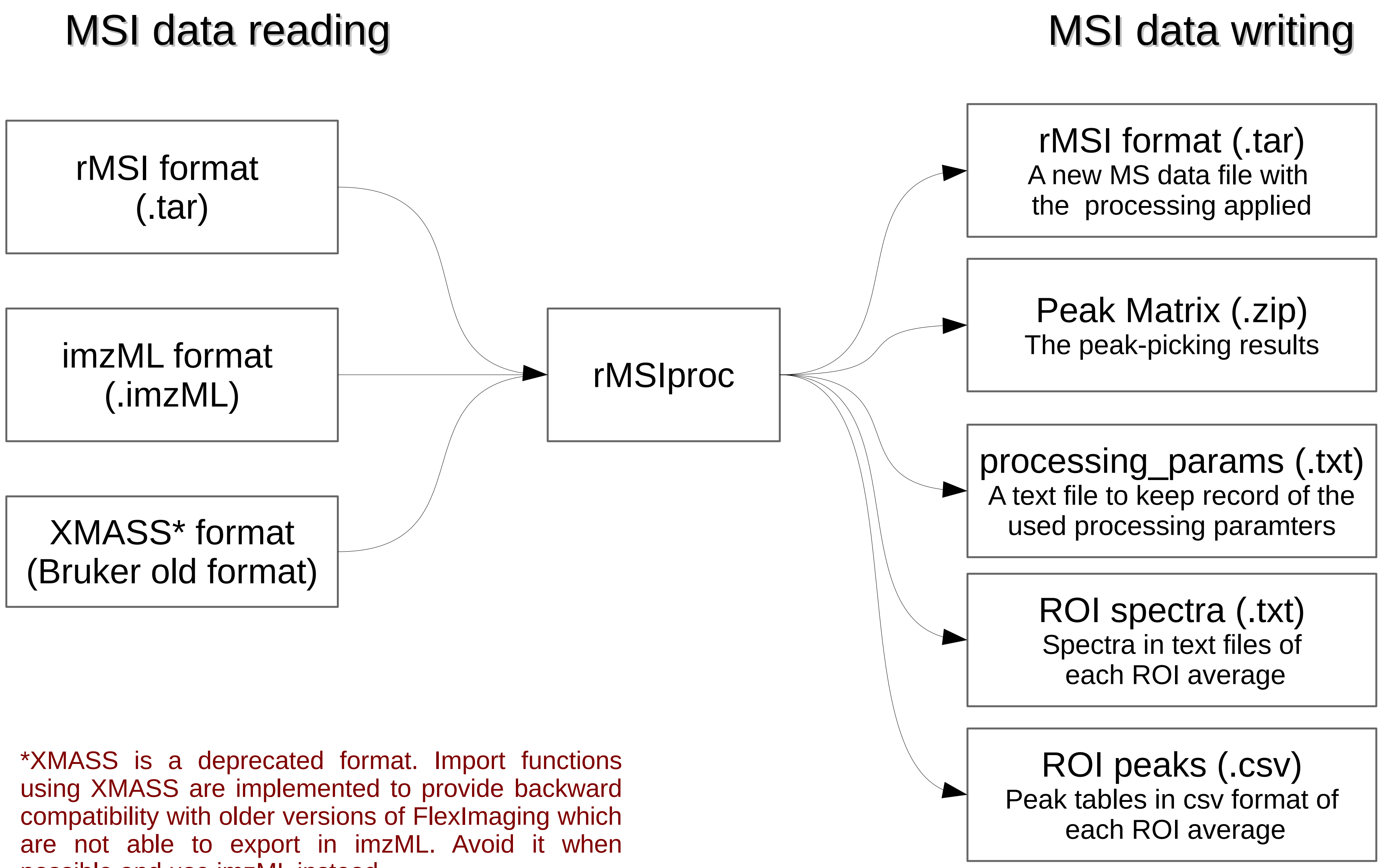
Processing Threads
Max Threads: 8

✓ Merged processing

Process Data

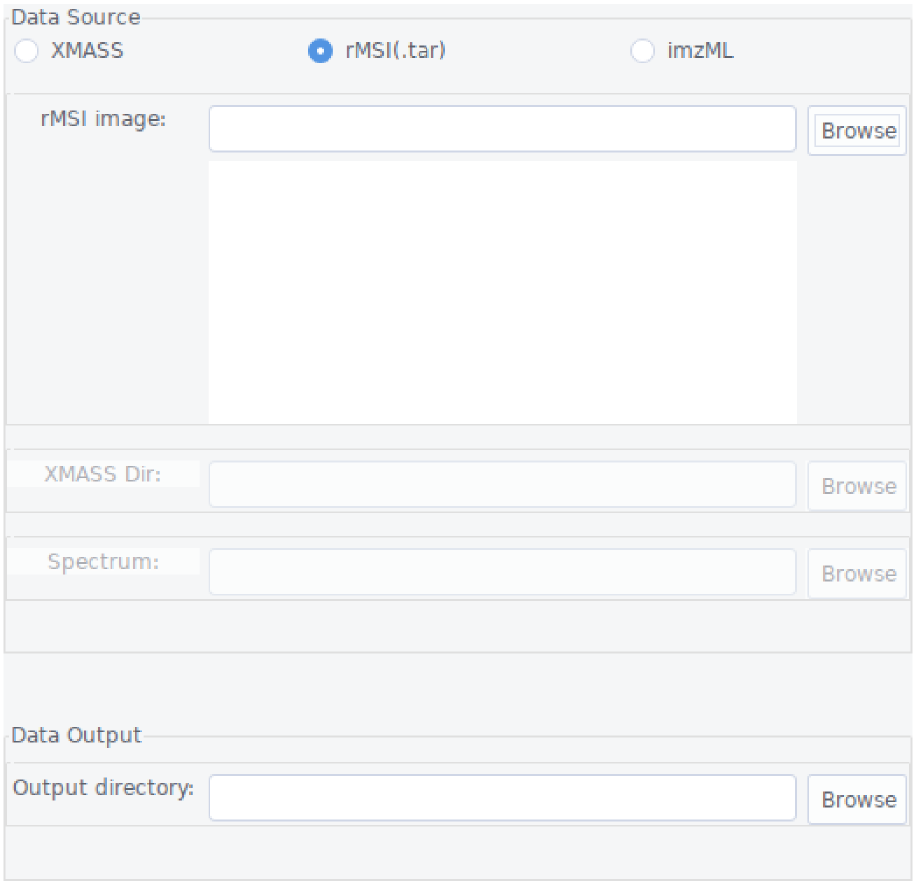
rMSIproc GUI

rMSIproc: Data input/output


MSI data reading
MSI data writing
rMSI format (.tar)
imzML format (.imzML)
XMASS* format (Bruker old format)
rMSIproc
rMSI format (.tar) A new MS data file with the processing applied
Peak Matrix (.zip) The peak-picking results
processing_params (.txt) A text file to keep record of the used processing paramters
ROI spectra (.txt) Spectra in text files of each ROI average
ROI peaks (.csv) Peak tables in csv format of each ROI average

*XMASS is a deprecated format. Import functions using XMASS are implemented to provide backward compatibility with older versions of FlexImaging which are not able to export in imzML. Avoid it when possible and use imzML instead.

rMSIproc parameters description



Data Source
XMASS [✓ rMSI(.tar)] imzML

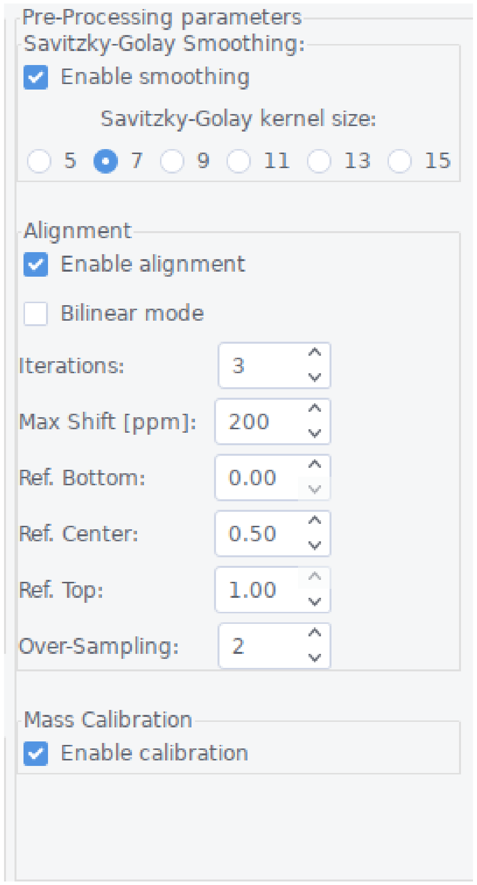
rMSI image:		Browse
XMASS Dir:		Browse
Spectrum:		Browse

Data Output

Output directory:		Browse

- Data Source: Select the input data format
- If XMASS is selected
  - XML path: Bruker ROI xml files
  - XMASS Dir: Data directory
  - Spectrum: A spectrum of a single pixel in text format. This is only used to obtain the mass axis from XMASS format properly.
- If rMSI (.tar) is selected
  - rMSI image: Select one or various MSI datasets to be processed.
- If imzML is selected
  - imzML image: Select one or various MSI datasets to be processed.
- Output directory: Directory where the processing result will be stored.

rMSIproc parameters description

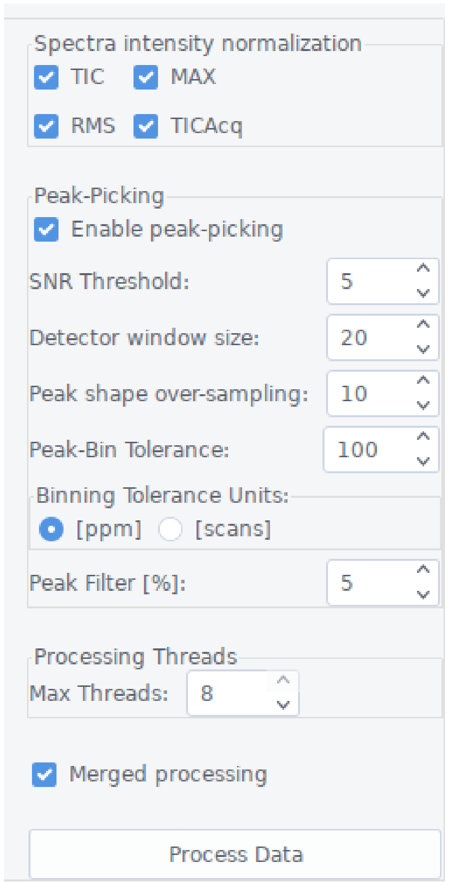

Pre-Processing parameters
Savitzky-Golay Smoothing:
[✓] Enable smoothing
Savitzky-Golay kernel size:
5 [✓ 7] 9 11 13 15

Alignment
[✓] Enable alignment
[ ] Bilinear mode
Iterations: 3
Max Shift [ppm]: 200
Ref. Bottom: 0.00
Ref. Center: 0.50
Ref. Top: 1.00
Over-Sampling: 2

Mass Calibration
[✓] Enable calibration

- Savitzky-Golay Smoothing
  - Enable smoothing: If enabled spectra will be smoothed
  - Savitzky-Golay kernel size: With higher value spectra will be more smoothed. Smoothing reduces noise but it increases the peak width as well.
- Alignment
  - Enable alignment: If enabled spectra will be aligned
  - Bilinear mode: If enabled three references points will be calculated for the alignment. The bilinear mode provides a better accuracy at the middle part of the spectrum.
  - Iterations: Number of times to run the alignment routine for each spectrum. With more iterations the algorithm is more capable to compensate for non-linear misalignment. If bilinear mode is enabled, then a single iteration should be enough.
  - Max Shift [ppm]: Maximum shift in ppm that alignment is allowed to compensate. This is usefull to avoid shifting noisy spectra or uncorrelated data.
  - Ref. (bottom, center and top): From 0 to 1, at which positions of the mass axis the correlations must be maximized.
  - Over-sampling: The spectrum will be over-sampled for better alignment accuracy. A over-sampling of two means that the number of data points will be duplicated for the internal alignment routine.
- Mass Calibration: If enabled a GUI will be displayed for mass recalibration.

rMSIproc parameters description



Spectra intensity normalization
[✓] TIC [✓] MAX
[✓] RMS [✓] TICAcq

Peak-Picking
[✓] Enable peak-picking

SNR Threshold:	5
Detector window size:	20
Peak shape over-sampling:	10
Peak-Bin Tolerance:	100

Binning Tolerance Units:
[✓ [ppm]] [scans]

Peak Filter [%]:	5

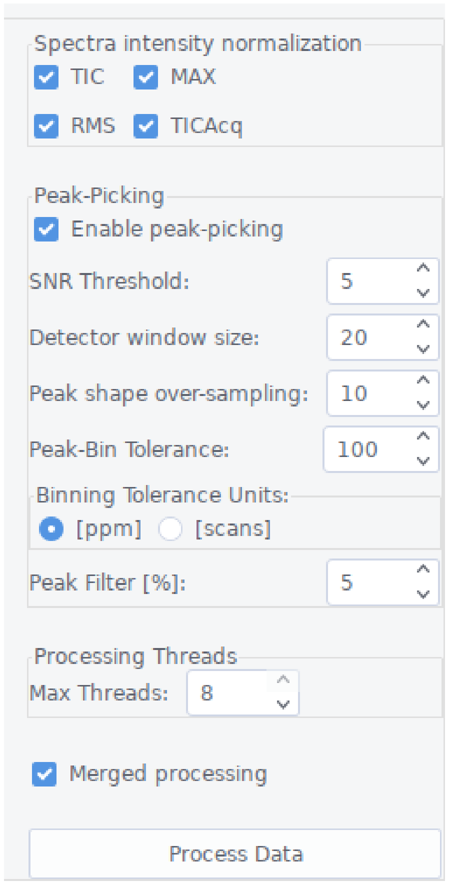
Processing Threads
Max Threads: 8

[✓] Merged processing

Process Data

- Spectra intensity normalization: Select the normalizations to apply. Normalizations coefficients will be stored in the $normalization field of the rMSI data objects and in the peak matrix. But, no normalization coefficient will be applied to the MS data. This allows using various normalizations coefficients since all of them are stored together with the processed data.
  - TIC: Sum of all spectrum intensities
  - MAX: The maximum of all intensities in each spectrum
  - RMS: Root mean square of spectrum intensities
  - TICAcq: TIC with a moving average windows applied in the same order as pixels were acquired. This normalization is designed to compensate for ionization source degradation during long time measurements

rMSIproc parameters description



Spectra intensity normalization

[✓ TIC] [✓ MAX]

[✓ RMS] [✓ TICAcq]

Peak-Picking

[✓ Enable peak-picking]

SNR Threshold: 5

Detector window size: 20

Peak shape over-sampling: 10

Peak-Bin Tolerance: 100

Binning Tolerance Units: [✓ [ppm]] [scans]

Peak Filter [%]: 5

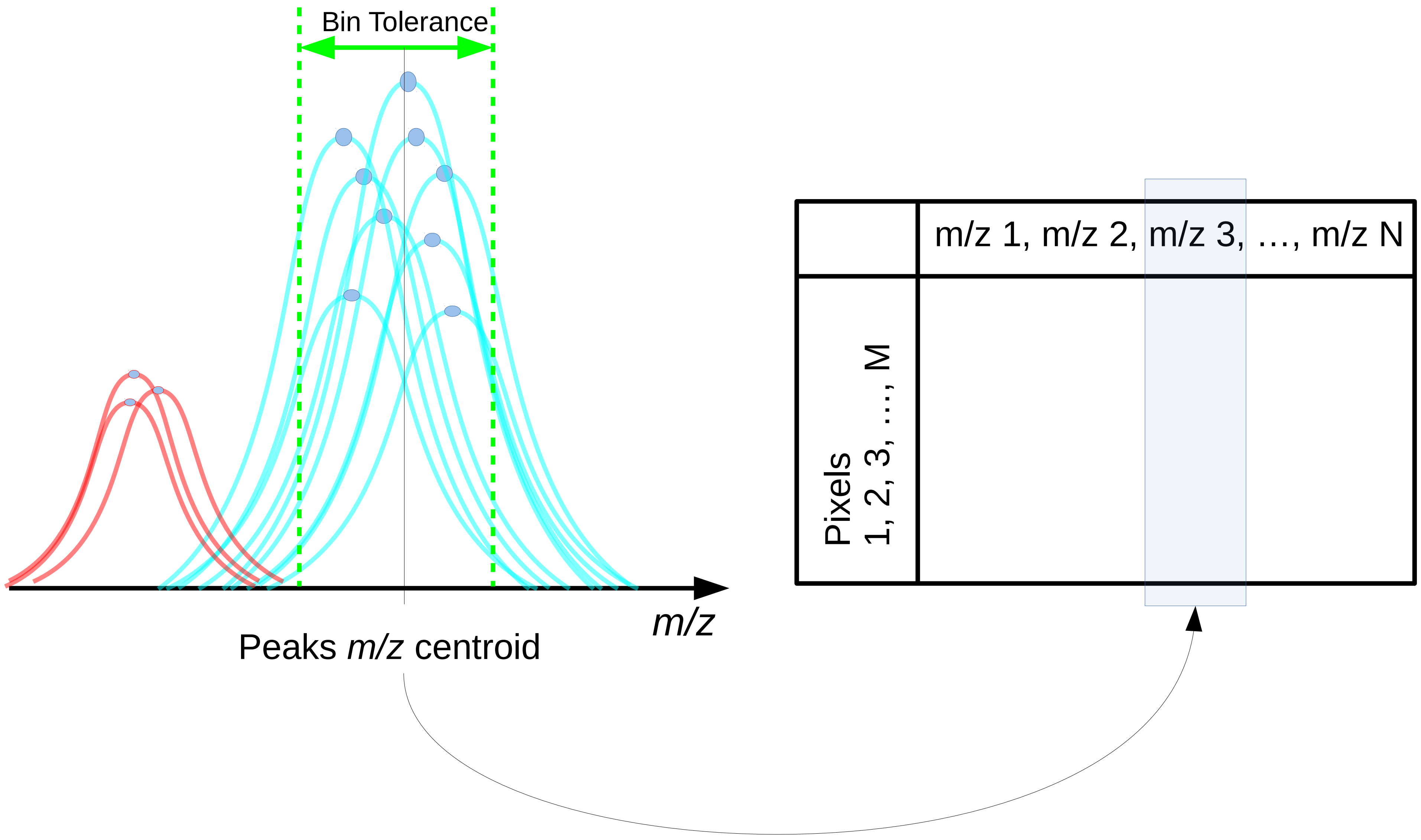
Processing Threads

Max Threads: 8

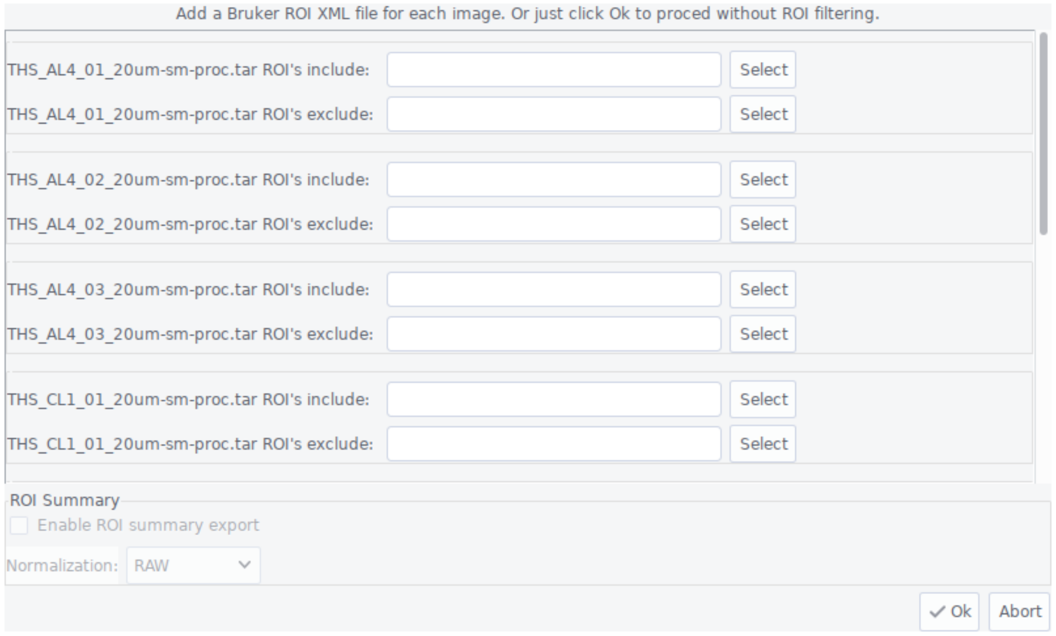
[✓ Merged processing]

- Peak-Picking
  - Enable peak-picking: If is enabled the peak matrix will be generated and stored.
  - SNR Threshold: Peaks with a signal to noise ratio below this parameter will be discarded.
  - Detector window size: The size of the moving window used to predict each peak mass. In general it is recommended to make it a bit higher than the number of data points in an intense peak.
  - Peak shape over-sampling: The over-sampling used for mass prediction. Low values produce worst peak mass accuracy but faster execution time. A value of 10 is a good balance for great accuracy and processing time.
  - Peak-Bin Tolerance: The tolerance used for the peak binning. Can be in ppm or in scans. Keep reading for further information.
  - Binning Tolerance Units: Can be specified in ppm or scans. This is applied after peak picking in order to merge all peaks related to the same compound in the same peak matrix column. If it is specified in ppm then a constant tolerance is applied to the whole mass range. If scans is used then the tolerance is relative the number of data points per peak.
  - Peak Filter [%]: If a peak mass was detected less times as specified in percent in this paramter, then this peak (matrix column) will be removed.
- Processing Threads: Specify how many CPU cores will be used.
- Merged processing: If more than one MSI dataset is selected it is possible to porcess all of them togheter in the same execution. This will produce a single peak matrix and a single calibration for all datasets.

The peak binning approach



rMSIproc ROI selection


Add a Bruker ROI XML file for each image. Or just click Ok to proced without ROI filtering.

THS_AL4_01_20um-sm-proc.tar ROI's include: Select
THS_AL4_01_20um-sm-proc.tar ROI's exclude: Select
THS_AL4_02_20um-sm-proc.tar ROI's include: Select
THS_AL4_02_20um-sm-proc.tar ROI's exclude: Select
THS_AL4_03_20um-sm-proc.tar ROI's include: Select
THS_AL4_03_20um-sm-proc.tar ROI's exclude: Select
THS_CL1_01_20um-sm-proc.tar ROI's include: Select
THS_CL1_01_20um-sm-proc.tar ROI's exclude: Select

ROI Summary
Enable ROI summary export
Normalization: RAW

A form to input regions of interest for each MSI data set will be displayed after setting all processing parameters. If this form is kept unfilled and the the button “Ok” clicked, then no ROI will be used and each dataset will be processed completely.

Regions of interest can be supplied using Bruker’s flexImaging XML format. Up to two ROI files can be selected for each MSI dataset: include and exclude. All pixels listed in Bruker’s ROI’s include list will be processed. All pixels listed in Bruker’s ROI’s exclude list will not be processed even if the same pixels are specfied in an include ROI file.

If some ROI XML file is selected then the ROI Summary form will be available. This allows exporting ROI information as a short summary of the whole data. The normalization field allows selecting the desired normalization used to export the ROI summary. The ROI summary will contain the following data:

- An average spectrum in plain text format for each ROI
  - The resulting spectra can be open in software like mMass.
  - One txt file for each ROI will be generated
  - The selected intensity normalization will be applied

- A CSV formatted matrix with the average and standard deviation peak values of each ROI
  - Intensity matrix of ROI averages
  - Area matrix of ROI averages
  - SNR matrix of ROI averages
  - Can be imported in an Excel sheet

Working with an rMSIproc peak matrix

R code example: peak matrix

```r
# Once rMSIproc::ProcessWizard() is completed and a peak matrix is stored in
# a zip file, we can load the peaks in an R session using the following:
MyPeaks <- rMSIproc::LoadPeakMatrix(”/path/to/my/peakmatrix.zip”)

# We can obtain the masses of the peaks using MyPeaks$mass field.
# We can obtin peak intensity matrix using MyPeaks$intensity.
# Lets assume we did some statistics with the intensity matrix and we found that
# rows 2 and 10 are interesting, so we want to plot the spectra.
# First we load the MSI data using rMSI function
MyImg <- rMSI::LoadMsiData(”/path/to/my/MSIdataset.tar”)

# Now we must get the pixel ID’s corresponding to the matrix rows 2 and 10.
# As long as there is only one MSI dataset the rows are equal to the ID’s, however we can have
# the peaks of various datasets in the same peak matrix (data merge function).
# The best way to obtain the MSI dataset ID’s is using rMSIproc::getImgIdsFromPeakMatrixRows()
Idinfo <- rMSIproc::getImgIdsFromPeakMatrixRows(MyPeaks, c(2 , 10))

# Check that the MSI dataset name in Idinfo corresponds to MyImg name.
# An R which() function can also be used for that. (in future UUID test will be implemented)
# Then, we can obtain the MSI dataset ID’s:
MyID <- Idinfo[[1]]$id

# We load the spectra of those ID’s
MySpectra <- rMSI::loadImgChunkFromIds(MyImg, MyID)

# Then we can plot these two spectra labelling its peaks
rMSI::plotSpectra(mass = MyImg$mass, intensity = MySpectra[1, ], col = "red",
                     peaks_mass = MyPeaks$mass, peaks_intensity = MyPeaks$intensity[2,])

rMSI::plotSpectra(mass = MyImg$mass, intensity = MySpectra[2, ], col = "blue",
                     peaks_mass = MyPeaks$mass, peaks_intensity = MyPeaks$intensity[10,])
```

R help

Both packages contains build-in documentation using the R standard help.
This documentation can be browsed using the following commands:

```
    ??rMSI
    ??rMSIproc
```

The documentation of a specfic function can be read using the ? character:

```
    ?rMSI::LoadMsiData()
```

Or just pressing te F1 key over the function.