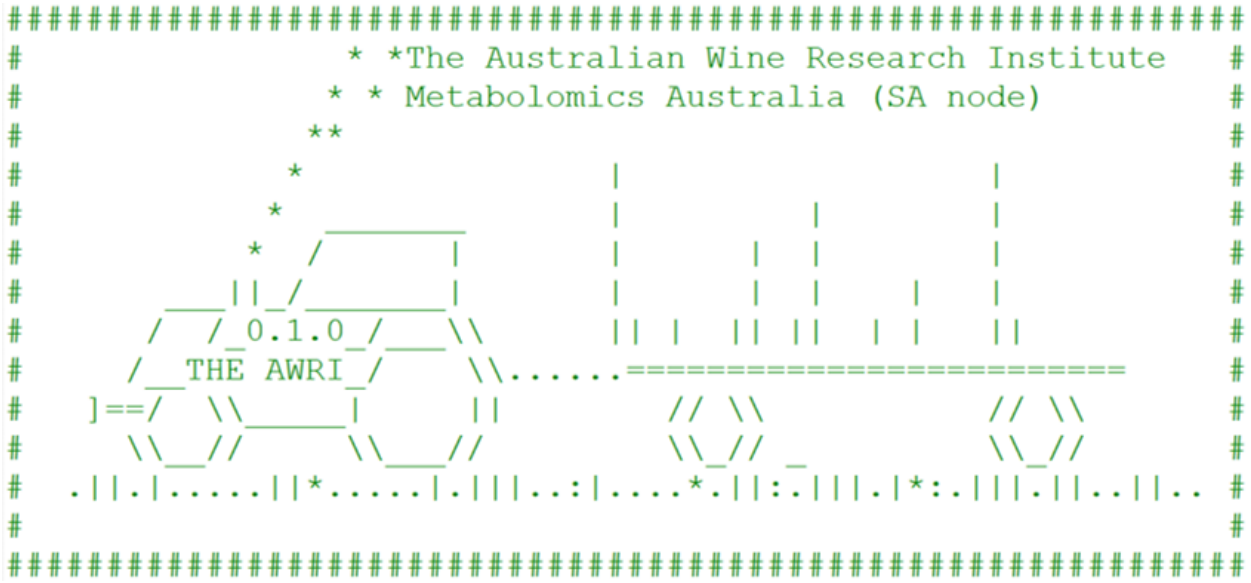
MStractor Workflow

Luca Nicolotti, The Australian Wine Research Institute, Metabolomics Australia

02/07/2020



Introduction

MStractor combines the functions for molecular feature extraction present in XCMS and CAMERA with user friendly dedicated GUIs for LC and MS parameter input, graphical QC outputs and descriptive statistics calculations.

A schematic representation of the workflow is displayed in **Figure 1**, reported below. The workflow consists of 10 steps with the possibility of running the workflow using either the most recent xcms functions specific for *XCMSnExp* object (available from xcms verison 1.51.5) (**branch a**) or the set of functions for *xcmsSet* objects (**branch b**). The 2 branches perform the same data processing steps and produce similar outputs, however, it is reccomende to use most recent xcms functions. Branch a is for a high computing power environment, whereas branch b is suitable for low computing power environments. Specifically, the authors observed that step **6a** tends to be quite time-consuming in particular whendealing with complex data set. Therefore, in a case where computing power is limited and the data set is large, the authors suggest using branch b of the workflow. It is ,however, reccomended to follow branch a and use the most updated xcms functions. Step 8 is common to both branches, where the data set is processed using CAMERA. Since the latest release available for CAMERA does not support XCMSnEXP objects, the data set object is required to be converted to the xcmsSet format prior to this step.

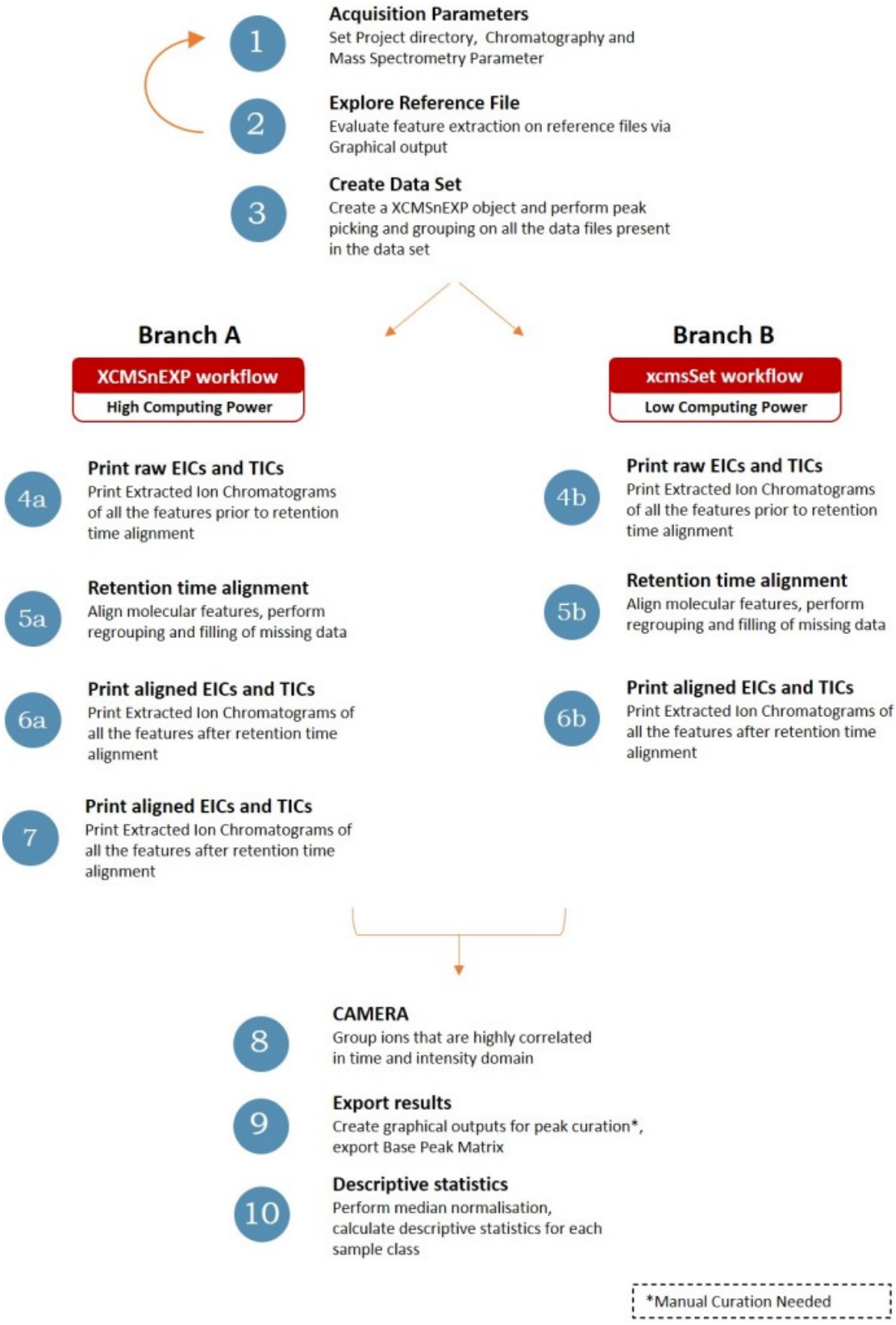

1
Acquisition Parameters
Set Project directory, Chromatography and Mass Spectrometry Parameter
2
Explore Reference File
Evaluate feature extraction on reference files via Graphical output
3
Create Data Set
Create a XCMSnEXP object and perform peak picking and grouping on all the data files present in the data set
Branch A
XCMSnEXP workflow
High Computing Power
Branch B
xcmsSet workflow
Low Computing Power
4a
Print raw EICs and TICs
Print Extracted Ion Chromatograms of all the features prior to retention time alignment
4b
Print raw EICs and TICs
Print Extracted Ion Chromatograms of all the features prior to retention time alignment
5a
Retention time alignment
Align molecular features, perform regrouping and filling of missing data
5b
Retention time alignment
Align molecular features, perform regrouping and filling of missing data
6a
Print aligned EICs and TICs
Print Extracted Ion Chromatograms of all the features after retention time alignment
6b
Print aligned EICs and TICs
Print Extracted Ion Chromatograms of all the features after retention time alignment
7
Print aligned EICs and TICs
Print Extracted Ion Chromatograms of all the features after retention time alignment
8
CAMERA
Group ions that are highly correlated in time and intensity domain
9
Export results
Create graphical outputs for peak curation*, export Base Peak Matrix
10
Descriptive statistics
Perform median normalisation, calculate descriptive statistics for each sample class
*Manual Curation Needed

Figure 1 : Schematic representation of MStractor Workflow

Each step of the framework displayed in Figure 1 involves using 1 or more functions. MStractor functions can be classified into 3 different groups:

1.MStractor specific functions: developed by the authors to provide the user with a friendly and seamless experience. The functions belonging to this class mainly involve the use of GUIs allowing parameter input and storage to be used later in the workflow. This group also include functions to normalise data and calculate descriptive statistics.

2.Wrapper functions: includes wrapper functions of xcms and CAMERA. The function backbone is the xcms (or CAMERA) function and additional code is used to automate routine operations, reduce parameter input errors and produce graphical outputs within the selected working directory.

3.xcms and CAMERA functions: these functions are basic CAMERA and xcms functions that are used along the workflow. However, the user is not required to update the function arguments because all the necessary parameters are input in the previous steps of the workflow

Data Preparation and File Naming

Preparing the data and creating the correct working directory and folder structure (outside the R environment) is required prior to executing the workflow. The raw data files need to be converted to an appropriate format which can be easily achieved using various tools freely available, such as Proteowizard (ProteWizard, http://proteowizard.sourceforge.net/) and Massmatrix (MassMatrix, http://www.massmatrix.net/). MStractor supported file formats include mzDATA, mzML and mzXML. After conversion, files should be grouped in folders according to their class of belonging. Every class-folder needs to be stored within the directory 'MSfiles'. The script will use the 'MSfiles' subfolder downstream to automatically determine the sample classes required for processing. If the folder 'MSfiles' is not present, the user will encounter errors. For each class-folder a minimum of 2 replicates is required. An example dataset (see Figure 2) is provided with the package.

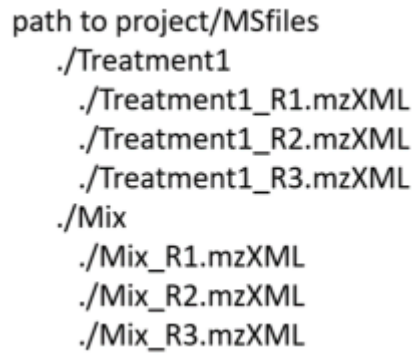


Figure 2 : Project subfolder structure

File naming is a key aspect for correct visualisation of graphical output downstream. A specific function of the workflow assigns different colours and symbols to the different sample classes. In this respect, the filename has to correspond to the folder name followed by an underscore and a number inidicating the biological replicate, as displayed in **Figure 2**.

Installation

To install the package from GitHub, make sure the package 'remotes' is installed and run the following,

```
library(remotes)

Sys.setenv(R_REMOTES_NO_ERRORS_FROM_WARNINGS="true")

remotes::install_github("MetabolomicsSA/MStractor")

# Alternatively, Download the tar.gz package from
# https://github.com/MetabolomicsSA/MStractor/releases
# and run the following

library(remotes)

Sys.setenv(R_REMOTES_NO_ERRORS_FROM_WARNINGS="true")

setRepositories(ind=1:2)

remotes::install_local("C:/pathtoPackage/MStractor_0.1.0.tar.gz",
    dependencies=NA)
```

After successful installation, load the MStractor library.

```
library(MStractor)
```

1. *Acquisition Parameters*

The first step of the workflow consists of running 6 functions that allow the input of chromatographic, mass spectrometry and peak picking parameters as well as loading the data and automatically defining the visualization settings for graphical QC outputs.

The default input values provided in thepackage functions are based on the settings of the acquisition instrument used by the authors and, therefore, suitable for processing the example data set available within the package. These values should be optimised for the user's LCMS system configuration, since they can dramatically influence the data processing outcome.

1.1 *Define Working Directory and Reference Files*

The function 'Project()' doesn't require arguments. Its execution opens a GUI (Figure 3) allowing the selection of the working directory and 2 reference files chosen from one of the folders within the MSfiles directory such as 2 pooled biological QC replicates(labeled as 'Mix' in the example data set). The function also creates a QC folder, that stores the graphical outputs to evaluate and QC input parameters at various steps of the workflow.

```
Project()
#If using the example dataset provided within the package set the
#working directory and click 'cancel' on the following 2 R-prompts asking
#to define refernce files
```

use the following lines of code to load the example dataset:

```
path<-system.file("extdata",package = "MStractor")
files <- dir(path, pattern = ".mzXML", full.names = TRUE)
```

Figure 3: GUI for project folder selection

1.2 *Input of Chromatographic and Mass Spectrometry Parameters*

ChromParam() allows the user to input the chromatographic parameters related to the analytical data set to be processed and that will be stored and used in the later stages of the workflow. The values to be entered are the retention time range of the data set (rt start and rt end), the maximum retention time drift observed (rtDelta) and the minimum and maximum fullwidth at half maximun (FWHM min and max).

MassSpecParam() is mainly used to input mass spectrometry related parameters. The criteria to be entered are the acquisition mode (negative as default); the mass range to be considered (mz start and mz end); the number of expected charges (default value set at 3); the maximum number of chromatographic peaks (EICsMax) expected for a single EIC ,the sensitivity and the file type (default set to '.mzXML') .

Values are entered using GUIs as displayed in Figure 4.

Both the functions return the input parameters.

For more details about the parameters consult the xcms manual: https://www.bioconductor.org/packages/release/bioc/manuals/xcms/man/xcms.pdf)

```
#leave default values if using the example dataset included in the package

ChromParam()
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "max"
##
## [[3]]
## [1] 10
##
## [[4]]
## [1] 90
##
## [[5]]
## [1] 32
```

```
MassSpecParam()
```

```
## [[1]]
## [1] "negative"
##
## [[2]]
## [1] 100
##
## [[3]]
## [1] 1650
##
## [[4]]
## [1] 0.01
##
## [[5]]
## [1] 3
##
## [[6]]
## [1] 30
##
## [[7]]
## [1] 0.7
##
## [[8]]
## [1] ".mzXML"
```

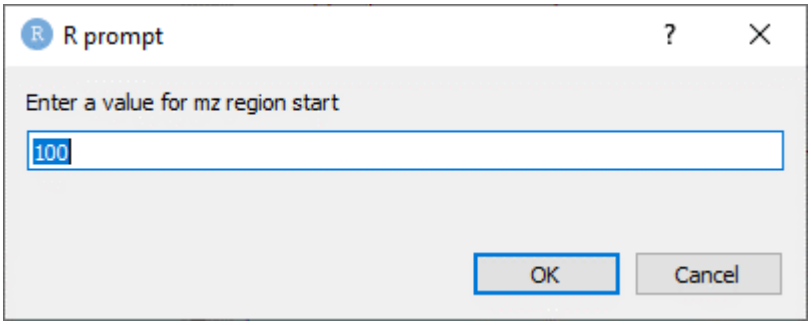


Figure 4: Example of GUI input parameters

1.3 *Load the Dataset*

The function returns a list of the loaded files. It creates a timestamp tracking the amount of time to complete the data processing and creates the object 'ClassType' which associates every data file with the class of belonging.

```
#Skip this step if using the example dataset provided within the package
LoadData()
```

1.4 *Define Class Identifiers*

The argument of the function is the object 'ClassType'. The function defines and returns specific symbols and colours for each sample class, which are then used to produce graphical outputs.

```
## don't run if using the the provided example dataset
```

```
ClassType<-c('Mix','Treatment1')

DefineClassAttributes(ClassType)
```

```
## [[1]]
## [1] "Mix"        "Mix"        "Mix"        "Treatment1" "Treatment1"
## [6] "Treatment1"
##
## [[2]]
## [1] "#FF0000" "#FF0000" "#FF0000" "#00FFFF" "#00FFFF" "#00FFFF"
##
## [[3]]
## [1] 1 1 1 2 2 2
```

1.5 *Define Peak Picking Parameters*

The function uses GUIs to define xcms peak picking parameters using the centwave algorithm. These include: minimum and maximum peakwidth, minimum and maximum m/z error (in ppm), values for the integration threshold (set at 2000) and signal to noise threshold. The pick picking can be set to gaussian and the integration method can be 1 or 2, as for the xcms documentation. The default values for minimum and maximum m/z error and signal to noise (S/N) threshold are set to 'none' and correspond, respectively, to 3.03 and 50 ppm and to 1000 for S/N value. Detailed information about the mentioned parameters can be found in the xcms documentation. https://www.bioconductor.org/packages/ release/bioc/manuals/xcms/man/xcms.pdf

```
#leave default values if using the package dataset
PeakPickingParam()
```

```
## [[1]]
## [1] 10
##
## [[2]]
## [1] 20
##
## [[3]]
## [1] 3.030303
##
## [[4]]
## [1] 50
##
## [[5]]
## [1] 26.51515
##
## [[6]]
## [1] 0.002
##
## [[7]]
## [1] FALSE
##
## [[8]]
## [1] 1
```

2. *Parameter Evaluation*

This is a QC step that allows testing the suitability of the input parameters on the reference files selected in the first step.

First, a GUI is used to define the settings of xcms functions findChromPeaks and groupChromPeaks which perform peak picking and grouping of molecular features. Secondly, data files are read using the xcms function readMSData and, lastly, peak picking and grouping is carried out. The output of the function is the XCMSnEXP object 'x_refs'.

```
#not to be executed if using the example  dataset
exploreRefs()
```

Note: In case the example dataset is used, the following needs to be run.

```
dir.create("./QC")
SampleGroup<-c("Mix", "Mix")
symbol<-c(1,1)
ClassCol<-c("#FF0000FF", "#FF0000FF")
path<-system.file("extdata",package = "MStractor")
files <- dir(path, pattern = ".mzXML", full.names = TRUE)
test<-files[1:2]
pd <- data.frame(sample_name = sub(basename(test), pattern = filetype,
    replacement ="", fixed = TRUE),sample_group = SampleGroup,
    stringsAsFactors = FALSE)
ref_data <- readMSData(test, pdata = new("NAnnotatedDataFrame",
    pd), mode = "onDisk")
expRefData(ref_data)
```

The refTic function returns a plot of the overlaid TICs for the reference files

```
refTic(x_refs)
```

```
## null device
##           1
```

Figure 5: Non-aligned overlaid TICs

The get100 function randomly picks 100 features detected across the retention time domain and plots them into a matrix. Using this output, it is easy to check whether all the features are correctly integrated.

```
get100(x_refs)
```

Figure 6: EICs of 100 features randomly picked across the retention time

3. *Create XCMSnExp Dataset*

An XCMSnEXP dataset containing all the raw files to be processed is automatically created based on the folder structure defined in
the data preparation step. The function performs the same steps described for exploreRefs(), with the only difference that the processing is applied to the whole dataset.

It is important that the input parameters used in this step match the ones defined for the reference files. After performing peak picking on each datafile,peaks are matched across all samples and grouped using the xcms functions findChromPeaks and groupChromPeaks. The results are stored in the XCMSnEXP object 'xdata'

```
peakPickGroup() # don't run if using the example dataset
```

In case the example dataset is used, the following needs to be run

```
ClassType<-c('Mix', 'Treatment1')
SampleGroup<-c("Mix",        "Mix",        "Mix",
        "Treatment1", "Treatment1", "Treatment1")
symbol<-c(1 ,1, 1, 2,2, 2)
ClassCol<- c("#FF0000FF", "#FF0000FF", "#FF0000FF",
        "#00FFFFFF", "#00FFFFFF", "#00FFFFFF")
path<-system.file("extdata",package = "MStractor")
files <- dir(path, pattern = ".mzXML", full.names = TRUE)
pd <- data.frame(sample_name = sub(basename(files), pattern = filetype,
    replacement = "", fixed = TRUE), sample_group = SampleGroup,
    stringsAsFactors = FALSE)
raw_data <- readMSData(files =files, pdata = new("NAnnotatedDataFrame",
    pd), mode = "onDisk")
ppgExData(raw_data)
```

Workflow Branching

The workflow forks at step 4 in branch a and branch b, as displayed in **Figure 1**. From this point on, the user can choose to either proceed with the XCMSnEXP object or perform a conversion into a xcmsSet object. Workflow branch A requires higher computing power than branch b. **Branches a** and **b** should ot be run together.

Branch A

4a *Print raw EICs and TICs*

Step 4 produces graphical outputs that can be used to visualise macroscopic differences among chromatographic profiles. For both branches plots of overlaid TIC (**Figure7**) chromatograms and for individual EICs are stored in the QC directory.

```
OverlaidTICs(xdata, 'raw') #raw indicates non-retention time aligned signals
printEICs(xdata, 'raw')
```

Figure 7: Overlaid TICs

5a *Retention time alignment, grouping and peak filling*

Retention time correction parameters are entered via GUI. The retention time alignment method currently supported is 'loess' (see xcms manual), while Obiwarp will be possibly implemented in the future MStractor releases. A plot of the retention time deviation is also generated and stored in the QC folder. After correction, features are regrouped and the filling of missing signals is performed.

```
RTalign(xdata,'loess')

xdata <- groupChromPeaks(xdata, param = PeakDensityParam(sampleGroups =
        xdata$sample_group, minFraction = 0.3, bw = 20))

xfilled <- fillChromPeaks(xdata, param =(FillChromPeaksParam(ppm = 50,
        expandMz = 0.5)))
```

6a *Print aligned EICs and TICs*

Step 6 replicates the series of functions already described for step 4a, with the only difference that retention time aligned TICs and EICs are generated

```
OverlaidTICs(xdata, 'aligned')
printEICs(xfilled, 'filled')
#'aligned' and 'filled' indicate retention time aligned signals
```

7a *Create Summary Datamatrix*

Step 7 is specific for **branch a** and collates feature information generated by the xcms featureDefinition() function with their corresponding response (either the integrated peak area 'into' or maximum intensity'maxo'). The arguments of the function are the filled XCMSnEXP object and the type of instrumental response desired (either 'into' or 'maxo'). After running step seven, the workflow merges again and the user has to proceed with step 8.

```
CreateDM(xfilled,'maxo')
```

Following this, the xsetConvert function (already described in step 4) reverts the XCMSnEXP object into the xcmsSet one. This is necessary for using CAMERA in the subsequent steps.

```
xsetConvert(xfilled)

## Note: you might want to set/adjust the 'sampclass' of the returned xcmSet object before proceeding w

sampnames(xset)<-spn
```

Branch B

The user needs to revert the object to a xcmsSet using the following code. This is required to execute function listed in the remaining sections of this document.

```
xsetConvert(xdata)
sampnames(xset)<-spn
```

4b *Print raw EICs and TICs*

```
getTICs(xcmsSet= xset, pngName= "./QC/TICs_raw.png", rt= "raw")
#raw indicates non-retention time aligned signals
printEICsXset(xset,'raw')
```

5b *Retention time alignment, grouping and peak filling*

Retention time correction parameters are entered via GUI. The retention time alignment method currently supported is 'loess' (see xcms manual), while Obiwarp will be implemented in the future releases. A plot of the retention time deviation is also generated and stored in the QC folder. After correction, features are regrouped and the filling of missing signals is performed.

```
RTalign_xset(xset,'loess')
xsAlign <- group(xsAlign, method= "nearest", mzVsRTbalance= 10, mzCheck=
        mzErrAbs,rtCheck= rtDelta, kNN=10)
xsFilled <- fillPeaks(xsAlign, method="chrom", expand.mz=0.5)
```

6b *Print aligned EICs and TICs*

Step 6 replicates the series of functions already described for stage 4, with the only difference that retention time aligned TICs and EICs are generated

```
getTICs(xcmsSet= xsAlign, pngName= "./QC/TICs_Aligned.png", rt= "corrected")
printEICsXset(xsFilled,'corrected')
# 'corrected' indicates retention time aligned signals
```

Workflow Merging

8. *Spectra Reconstruction*

From step 8, the workflow is the same for branches **a** and **b**. Spectra reconstruction is carried out using the function autoCAMERA()that includes the 3 CAMERA functions groupFWHM(), findIsotopes() and groupCorr(). A GUI allows the input of the function parameters. Consult CAMERA manual for more information: https://www.bioconductor.org/packages/release/bioc/manuals/CAMERA/man/ CAMERA.pdf. Using autoCamera, the peak table embedded within the xcmsSet object is extracted and features arranged into pseudospectra groups according to their retention times. Then, based on the maximum expected charge test, isotopic patterns are located and the features belonging to coeluting compounds are resolved using a correlation matrix.

```
#branch A
xset

## An "xcmsSet" object with 6 samples
##
## Time range: 11.7-8505 seconds (0.2-141.8 minutes)
## Mass range: 96.9573-1565.3684 m/z
## Peaks: 15284 (about 2547 per sample)
## Peak Groups: 2242
## Sample classes: Mix, Treatment1
##
## Feature detection:
##  o Peak picking performed on MS1.
##  o Scan range limited to  1 - 4400
## Profile settings: method = bin
##                   step = 0.1
##
## Memory usage: 6.81 MB

autoCamera(xset)

## Start grouping after retention time.
## Created 649 pseudospectra.
## Generating peak matrix!
## Run isotope peak annotation
##  % finished: 10  20  30  40  50  60  70  80  90  100
## Found isotopes: 310
## Start grouping after correlation.
## Generating EIC's ..
## Warning: Found NA peaks in selected sample.
```

```
##
## Calculating peak correlations in 649 Groups...
##  % finished: 10  20  30  40  50  60  70  80  90  100
##
## Calculating peak correlations across samples.
##  % finished: 10  20  30  40  50  60  70  80  90  100
##
## Calculating isotope assignments in 649 Groups...
##  % finished: 10  20  30  40  50  60  70  80  90  100
## Calculating graph cross linking in 649 Groups...
##  % finished: 10  20  30  40  50  60  70  80  90  100
## New number of ps-groups:  1054
## xsAnnotate has now 1054 groups, instead of 649
```

```
#branc B
#autoCamera(xsFilled)
```

The function generates the object PksAn, a datamatrix containing the results of the spectra reconstruction.

9. *Export Matrix*

Step 9 provides a series of 4 functions that generate the final results of the workflow. The second argument of the function "collectBP_EICs" is either "filled" for workflow branch a or "corrected" for workflow branch b

By using FilterDM() pseudospectra containing less than a fixed number of ions (2 by default) are removed from the data matrix. Then, only the most intense ion for each pseudospectrum is retained (base peak response). This function was implemented to obtain a simplified data matrix avoiding redundant information and allowing for quick and efficient downstream data processing.

The output is the BasePks object, a matrix summarizing the results of data filtering.

CollectBP_EICs() prints the extracted ion chromatograms related to the base peak matrix in a '.png' format. The function also returns the time required for processing the dataset. The png files are duplicated into 2 directories, named **'EICs_BasePeaks'** and **'EICs_BasePeaks_Curated'**. The former is used as data back-up, while the latter allows the user to perform a final curation on the EICs, by removing those that do not contain useful information. This represents the only step of the entire workflow where a manual input of the user is required. An example is displayed in **Figure 8**.

BasePks_Curated() generates an updated data matrix (BasePksCur) with the discarded features removed. The final matrix is saved in a '.tsv' file called 'Pks,BPs_Curated' which is saved within the working directory.

Lastly, by using MedianNormalize() the curated matrix is normalized on the median value. This is done to minimize possible minimal run-to-run variation in the instrument performances. The output is saved in the .tsv file 'Normalized Matrix'.

```
FilterDM(PksAn, xset)
```

```
#the second argument of the function below is either 'filled' (branch a)
#or 'corrected' (branch b)
CollectBP_EICs(BasePks,'filled')
```

```
## Time difference of 55.55243 mins
```

```
#after this step manual curation is necessary
BasePks_Curated(BasePks)
MedianNormalize(BasePksCur, xset)
```

Figure 8: Curation example

10. *Descriptive Statistics*

An additional step of the workflow allows calculating descriptive statistics (average value, standard deviation and % CV) on the analytical replicates of each sample class. The output is stored in separate .tsv file for each class.

```
StatsByClass(ClassType, xset)
```

Session information

```
sessionInfo()
```

```
## R version 4.0.2 (2020-06-22)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19041)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_Australia.1252  LC_CTYPE=English_Australia.1252
## [3] LC_MONETARY=English_Australia.1252 LC_NUMERIC=C
## [5] LC_TIME=English_Australia.1252
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] MStractor_0.1.0     xcms_3.11.3         MSnbase_2.15.2
##  [4] ProtGenerics_1.21.0 S4Vectors_0.27.12   mzR_2.23.0
##  [7] Rcpp_1.0.4.6        BiocParallel_1.23.0 Biobase_2.49.0
## [10] BiocGenerics_0.35.4
##
## loaded via a namespace (and not attached):
##   [1] colorspace_1.4-1            ellipsis_0.3.1
##   [3] htmlTable_2.0.0             XVector_0.29.2
##   [5] GenomicRanges_1.41.5        base64enc_0.1-3
##   [7] rmutil_1.1.5                clue_0.3-57
##   [9] rstudioapi_0.11             affyio_1.59.0
##  [11] statip_0.2.3                codetools_0.2-16
##  [13] splines_4.0.2               ncdf4_1.17
##  [15] doParallel_1.0.15           impute_1.63.0
##  [17] robustbase_0.93-6           knitr_1.29
##  [19] Formula_1.2-3               cluster_2.1.0
##  [21] vsn_3.57.0                  png_0.1-7
##  [23] stabledist_0.7-1            graph_1.67.1
```

```
##  [25] BiocManager_1.30.10         compiler_4.0.2
##  [27] backports_1.1.8             Matrix_1.2-18
##  [29] limma_3.45.7                acepack_1.4.1
##  [31] htmltools_0.5.0             tools_4.0.2
##  [33] igraph_1.2.5                modeest_2.4.0
##  [35] gtable_0.3.0                glue_1.4.1
##  [37] GenomeInfoDbData_1.2.3      affy_1.67.0
##  [39] RANN_2.6.1                  dplyr_1.0.0
##  [41] MALDIquant_1.19.3           vctrs_0.3.1
##  [43] multtest_2.45.0             preprocessCore_1.51.0
##  [45] iterators_1.0.12            timeDate_3043.102
##  [47] xfun_0.15                   stringr_1.4.0
##  [49] spatial_7.3-12              lifecycle_0.2.0
##  [51] gtools_3.8.2                XML_3.99-0.3
##  [53] DEoptimR_1.0-8              zlibbioc_1.35.0
##  [55] MASS_7.3-51.6               scales_1.1.1
##  [57] timeSeries_3062.100         pcaMethods_1.81.0
##  [59] SummarizedExperiment_1.19.5 RBGL_1.65.0
##  [61] MassSpecWavelet_1.55.0      RColorBrewer_1.1-2
##  [63] yaml_2.2.1                  gridExtra_2.3
##  [65] ggplot2_3.3.2               fBasics_3042.89.1
##  [67] rpart_4.1-15                latticeExtra_0.6-29
##  [69] stringi_1.4.6               foreach_1.5.0
##  [71] checkmate_2.0.0             stable_1.1.4
##  [73] svGUI_1.0.0                 GenomeInfoDb_1.25.2
##  [75] svDialogs_1.0.0             rlang_0.4.6
##  [77] pkgconfig_2.0.3             matrixStats_0.56.0
##  [79] bitops_1.0-6                mzID_1.27.0
##  [81] evaluate_0.14               fda_5.1.4
##  [83] lattice_0.20-41             purrr_0.3.4
##  [85] htmlwidgets_1.5.1           tidyselect_1.1.0
##  [87] plyr_1.8.6                  magrittr_1.5
##  [89] R6_2.4.1                    IRanges_2.23.10
##  [91] snow_0.4-3                  generics_0.0.2
##  [93] Hmisc_4.4-0                 DelayedArray_0.15.5
##  [95] sm_2.2-5.6                  pillar_1.4.4
##  [97] foreign_0.8-80              berryFunctions_1.19.1
##  [99] nnet_7.3-14                 survival_3.2-3
## [101] abind_1.4-5                 RCurl_1.98-1.2
## [103] tibble_3.0.1                CAMERA_1.45.1
## [105] crayon_1.3.4                rmarkdown_2.3
## [107] jpeg_0.1-8.1                grid_4.0.2
## [109] data.table_1.12.8           digest_0.6.25
## [111] tidyr_1.1.0                 munsell_0.5.0
## [113] tcltk_4.0.2
```

References

1)Smith, C.A. and Want, E.J. and O'Maille, G. and Abagyan,R. and Siuzdak, G.: XCMS:Processing mass spectrometry data for metabolite profiling using nonlinear peak alignment, matching and identification, Analytical Chemistry, 78:779-787 (2006)

2)Ralf Tautenhahn, Christoph Boettcher, Steffen Neumann: Highly sensitive feature detection for high resolution LC/MS BMC Bioinformatics, 9:504 (2008)

3)H. Paul Benton, Elizabeth J. Want and Timothy M. D. Ebbels Correction of mass calibration gaps in liquid chromatography-mass spectrometry metabolomics data Bioinformatics, 26:2488 (2010)

4)Kuhl, C., Tautenhahn, R., Boettcher, C., Larson, T. R. and Neumann, S. CAMERA: an integrated strategy for compound spectra extraction and annotation of liquid chromatography/mass spectrometry data sets. Analytical Chemistry, 84:283-289 (2012)