

# Supplementary data for: ‘MSnbase’, efficient and elegant R-based processing and visualisation of raw mass spectrometry data

***Laurent Gatto, Sebastian Gibb and Johannes Rainer***

**26 June 2020**

## Contents

1	Introduction . . . . .	2
2	Data handling and analysis with <b>MSnbase</b> . . . . .	2
2.1	Basic MS data access functionality . . . . .	3
2.2	Performance of the on-disk backend on large scale data sets . . . . .	4
2.3	System information . . . . .	7

## 1 Introduction

This document describes handling of mass spectrometry data from large experiments using the `MSnbase` package and more specifically its *on-disk* backend. For demonstration purposes, the `MassIVE` data set `MSV000080030` is used. This consists of over 1,000 mzXML files from swab-samples collected from hands and various personal objects of 80 volunteers.

## 2 Data handling and analysis with `MSnbase`

In this section we demonstrate data handling and access by `MSnbase` on a large experiment consisting of more than 1,000 data files.

To reproduce the analysis in this document, download the `MSV000080030` folder from <ftp://massive.ucsd.edu/MSV000080030/> and place it into the same folder than this document.

Below we load the required libraries and define the files to be analyzed.

```
library(MSnbase)
library(magrittr)
library(pryr)

fls <- dir("MSV000080030/ccms_peak/Forensic_study_80_volunteers/",
          pattern = "mzXML", full.names = TRUE, recursive = TRUE)
```

The data set consists of in total 1182 mzXML files. We next load the data using the two different `MSnbase` backends `"inMemory"` and `"onDisk"`. For the in-memory backend, due to the larger memory requirements, we import the data only from a subset of the files.

```
ms_mem <- readMSData(fls[grepl("Hand", fls)], mode = "inMemory")
```

Next we load data from all mzXML files as an on-disk `MSnExp` object.

```
ms_dsk <- readMSData(fls, mode = "onDisk")
```

Below we count the number of spectra per MS level of the whole experiment.

```
table(msLevel(ms_dsk))
##
##      1      2
## 1173678 4599786
```

Note that the in-memory `MSnExp` object contains only MS2 spectra (in total 2140520) from a subset of data files, still, data import was much slower (over ~ 12 hours for the in-memory backend while creating the on-disk object from the full data data set took ~ 3 hours).

Next we subset the on-disk object to contain the same set of spectra than the in-memory `MSnExp` and compare their memory footprint.

```
ms_dsk_hands <- ms_dsk %>%
  filterFile(grep("Hand", fls)) %>%
  filterMsLevel(2L)

object_size(ms_mem)
```

Supplementary data for: ‘MSnbase’, efficient and elegant R-based processing and visualisation of raw mass spectrometry data

```
## 21.8 GB
object_size(ms_dsk_hands)
## 617 MB
```

Since the on-disk object stores only spectra metadata in memory it occupies also much less system memory. As a comparison, the on-disk `MSnExp` for the full experiment was still much smaller than the in-memory object:

```
object_size(ms_dsk)
## 1.66 GB
```

## 2.1 Basic MS data access functionality

Before evaluating the `MSnbase` performance on the large data set we provide first some general description of the `MSnbase` data classes in `MSnbase` basic data access operations. MS data from raw data files in `mzML`, `mzXML`, `mzData` or `netCDF` format is represented by the `MSnExp` object which organizes the spectra from the original files in a one-dimensional list. Functions like `rtime` and `msLevel` allow to extract the retention time and MS level, respectively, for each spectrum and return a `numeric` (or `integer`) vector with the same length than the number of spectra in the `MSnExp`. In the example below we use the `rtime` function to extract the retention times for each spectrum.

```
rts <- rtime(ms_dsk)
length(rts)
## [1] 5773464
head(rts)
## F0001.S0001 F0001.S0002 F0001.S0003 F0001.S0004 F0001.S0005 F0001.S0006
##          0.470          0.803          1.136          1.468          1.801          2.134
```

The `fromFile` function can be used to determine from which file (sample) a specific spectrum in the `MSnExp` object is. This function returns an `integer` vector, same length than spectra in the experiment, with the file index; the file names can be accessed with the `fileNames` method. A `MSnExp` object can be subsetted with `[]` and e.g. the index of the spectra that should be retained. In the code block below we subset our `ms_dsk` object to contain only spectra from the 3rd file.

```
one_file <- ms_dsk[fromFile(ms_dsk) == 3]
length(one_file)
## [1] 4911
```

Note however that there are also dedicated *filter* functions to subset an `MSnExp` object such as `filterFile`, `filterMsLevel`, `filterRt`, `filterMz`, `filterPrecursorMz` or `filterIsolationWindow`. In the example below we use the `filterRt` function to further subset our data to contain only spectra acquired within a certain time range.

```
one_file <- filterRt(one_file, rt = c(40, 300))
length(one_file)
## [1] 1996
```

As mentioned above, the `MSnExp` object is comparable with a list of spectra. Thus, to extract a single spectrum from it we can use `[[`. This will return an object of type `Spectrum` which encapsulates/represents all information of the measured spectrum (i.e.  $m/z$  and intensity values as well as metadata information). In the example below we extract the 15th spectrum from our data subset and access its  $m/z$  values with the `mz` function.

```
sp <- one_file[[15]]
mz(sp)
## [1] 400.4412 431.2400 1617.8282
```

This particular spectrum has thus only 3 peaks.

Note that  $m/z$  or intensity values can also be directly extracted from the `MSnExp` object as shown in the example below. The result will be a `list` of `numeric` vectors, each element representing the  $m/z$  values for each spectrum in the object.

```
mzs <- mz(one_file)
class(mzs)
## [1] "list"
length(mzs)
## [1] 1996
```

In addition, it is also possible to extract all  $m/z$  and intensity values from an `MSnExp` object as a `data.frame` as shown in the code block below, but this is not suggested, since it loads all the data into memory and all MS spectrum metadata such as MS level or precursor  $m/z$  get lost.

```
df <- as(one_file, "data.frame")
head(df)
##   file    rt      mz    i
## 1    1 40.118 387.2650   88
## 2    1 40.118 389.2627  192
## 3    1 40.118 474.2964  164
## 4    1 40.450 387.2416  212
## 5    1 40.450 389.2666  184
## 6    1 40.450 445.2680  132
nrow(df)
## [1] 2854657
```

Note that for all these operations it is irrelevant whether an in-memory or on-disk backend was used. In general it is advisable to use the on-disk backend especially for experiments with more than  $\sim 50$  files.

## 2.2 Performance of the on-disk backend on large scale data sets

To demonstrate `MSnbase`'s efficiency in processing large scale experiments we perform some standard subsetting, data access and manipulation operations.

We first compare the performance of the on-disk and in-memory backend on accessing  $m/z$  values with the `mz` function on a set of 100 randomly selected spectra. The performance is assessed with the `microbenchmark` function.

Supplementary data for: 'MSnbase', efficient and elegant R-based processing and visualisation of raw mass spectrometry data

```
set.seed(123)
idx <- sample(seq_along(ms_mem), 100)

library(microbenchmark)
microbenchmark(mz(ms_mem[idx]),
               mz(ms_dsk_hands[idx]),
               times = 5)
## Unit: seconds
##           expr      min       lq     mean   median      uq      max
##  mz(ms_mem[idx]) 65.89336 67.43615 68.68761 67.46700 69.92259 72.71893
##  mz(ms_dsk_hands[idx]) 18.12663 24.56444 24.31474 24.73441 26.85985 27.28837
##  neval
##      5
##      5
```

Thus, for this combined subsetting and data access operation the on-disk backend performed better than the in-memory MSnExp, while even requiring much less memory.

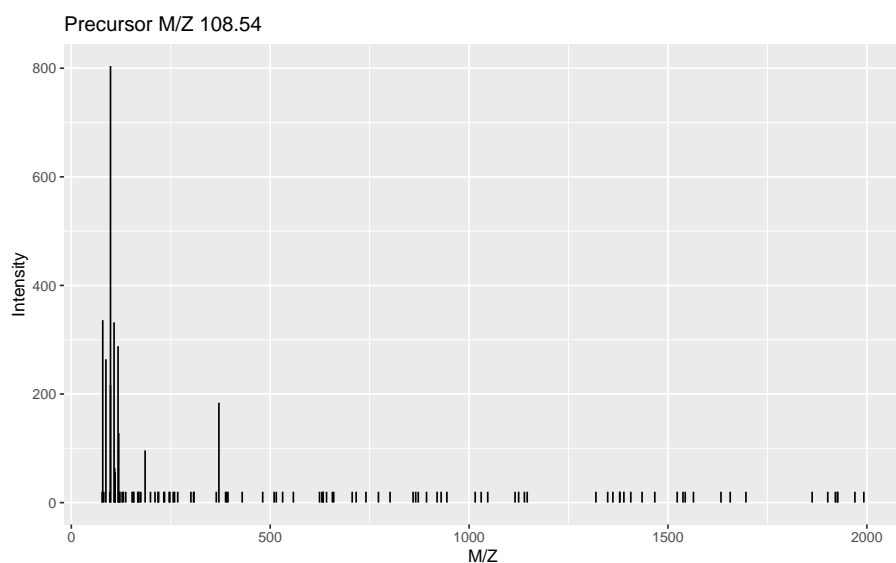
Next we extract all MS2 spectra with a retention time between 50 and 60 seconds and a precursor m/z of 108.5362 (+/- 5ppm). This subsetting operation is performed on the on-disk MSnExp object representing the full experiment with the 1182 data files/samples. To assess the performance of the following operations we use `system.time` calls that record elapsed time in seconds.

```
system.time(
  ms_sub <- ms_dsk %>%
    filterMsLevel(2L) %>%
    filterRt(c(50, 60)) %>%
    filterPrecursorMz(mz = 108.5362, ppm = 5)
)["elapsed"]
## elapsed
## 6.436
```

In total `length(ms_sub)` spectra were selected from in total 928 data files/samples. The plot below shows the data for the first spectrum.

```
system.time(
  plot(ms_sub[[1]])
)["elapsed"]
```

Supplementary data for: 'MSnbase', efficient and elegant R-based processing and visualisation of raw mass spectrometry data



```
## elapsed
## 0.672
```

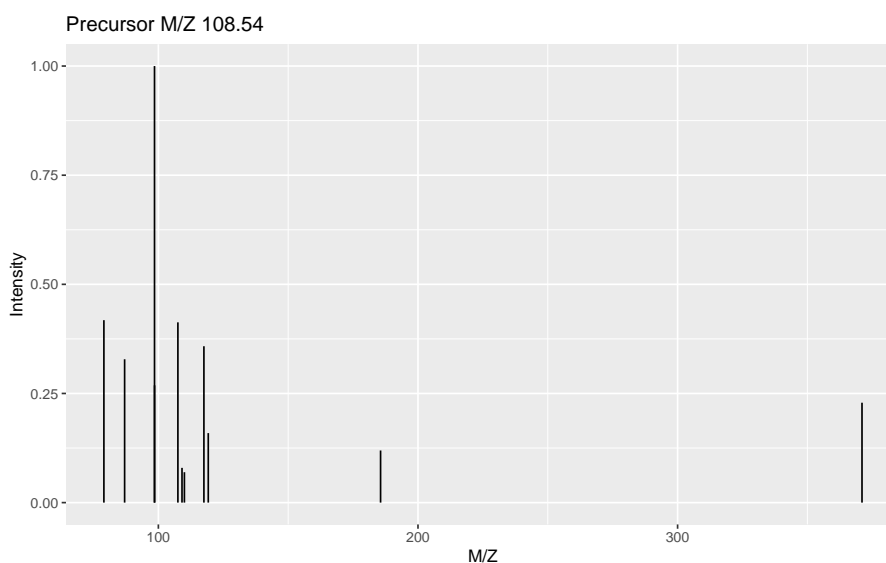
Since there seems to be quite some background noise in the MS2 spectrum we next remove peaks with an intensity below 50 by first replacing their intensities with 0 (with the `removePeaks` call) and subsequently removing all 0-intensity peaks from each spectrum with the `clean` call. In addition we *normalize* each spectrum by dividing the maximum intensity per spectrum from the spectrum's intensities.

```
system.time(
  ms_sub <- ms_sub %>%
    removePeaks(t = 50) %>%
    clean(all = TRUE) %>%
    normalize(method = "max")
)["elapsed"]
## elapsed
## 0.015
```

The result on the first spectrum is shown below.

```
system.time(
  plot(ms_sub[[1]])
)["elapsed"]
```

Supplementary data for: 'MSnbase', efficient and elegant R-based processing and visualisation of raw mass spectrometry data



```
## elapsed
## 0.414
```

Note that any of the data manipulations above are not directly applied to the data but *cached* in the object's internal *lazy processing queue* (explaining the very short running time of the normalization call). The operations are only effectively applied to the data when m/z or intensity values are extracted from the object as it is the case in the `plot` call above.

For additional workflows employing MSnbase see also [metabolomics2018](https://github.com/jorainer/metabolomics2018)<sup>1</sup> that explains filtering, plotting and centroiding of profile-mode MS data with MSnbase and subsequent pre-processing of the (label free/untargeted) LC-MS data with the `xcms` package (that builds upon MSnbase for MS data representation and access).

<sup>1</sup><https://github.com/jorainer/metabolomics2018>

## 2.3 System information

The present analysis was run on a MacBook Pro 16,1 with 2.3 GHz 8-Core Intel Core i9 CPU and 64 GB 2667 MHz DDR4 memory running macOS version 10.15.5. The R version and the version of the used packages are listed below.

```
sessionInfo()
## R version 4.0.0 (2020-04-24)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Catalina 10.15.5
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats4 parallel stats graphics grDevices utils datasets
```

Supplementary data for: 'MSnbase', efficient and elegant R-based processing and visualisation of raw mass spectrometry data

```
## [8] methods base
##
## other attached packages:
## [1] microbenchmark_1.4-7 BiocParallel_1.22.0 pryr_0.1.4
## [4] magrittr_1.5 MSnbase_2.14.2 ProtGenerics_1.20.0
## [7] S4Vectors_0.26.1 mzR_2.22.0 Rcpp_1.0.4.6
## [10] Biobase_2.48.0 BiocGenerics_0.34.0 BiocStyle_2.16.0
## [13] rmarkdown_2.3
##
## loaded via a namespace (and not attached):
## [1] tidyselect_1.1.0 xfun_0.15 purrr_0.3.4
## [4] lattice_0.20-41 colorspace_1.4-1 vctrs_0.3.1
## [7] generics_0.0.2 htmltools_0.5.0 yaml_2.2.1
## [10] vsn_3.56.0 XML_3.99-0.3 rlang_0.4.6
## [13] pillar_1.4.4 glue_1.4.1 affy_1.66.0
## [16] foreach_1.5.0 affyio_1.58.0 lifecycle_0.2.0
## [19] plyr_1.8.6 mzID_1.26.0 stringr_1.4.0
## [22] zlibbioc_1.34.0 munsell_0.5.0 pcaMethods_1.80.0
## [25] gtable_0.3.0 codetools_0.2-16 evaluate_0.14
## [28] labeling_0.3 knitr_1.29 IRanges_2.22.2
## [31] doParallel_1.0.15 preprocessCore_1.50.0 scales_1.1.1
## [34] BiocManager_1.30.10 limma_3.44.3 farver_2.0.3
## [37] impute_1.62.0 ggplot2_3.3.2 digest_0.6.25
## [40] stringi_1.4.6 bookdown_0.20 dplyr_1.0.0
## [43] ncd4_1.17 grid_4.0.0 tools_4.0.0
## [46] tibble_3.0.1 crayon_1.3.4 pkgconfig_2.0.3
## [49] ellipsis_0.3.1 MASS_7.3-51.6 iterators_1.0.12
## [52] R6_2.4.1 MALDIquant_1.19.3 compiler_4.0.0
```