Direct access to mass spectrometry data in R - The `MsBackendRawFileReader`

Tobias Kockmann, Christian Panse

Functional Genomics Center Zurich, Swiss Federal Institute of Technology Zurich | University of Zurich, Winterthurerstr. 190, CH-8057 Zurich, SWITZERLAND.

Abstract

The R for Mass Spectrometry (RforMS) initiative aims to provide efficient, thoroughly documented, tested and flexible software for the analysis of mass spectrometry data in R (`https://www.rformassspectrometry.org`). Reaching this goal at first requires performant and genuine access to the data from within the R environment. Unfortunately, proprietary vendor formats have so far obstructed this direct access. Our R package **rawR** (source: `https://github.com/cpanse/MsBackendRawFileReader`) eliminates this limitation for Thermo Fisher Scientific raw files by wrapping methods defined in the RawFileReader .NET assembly [1]. The resulting on-disc `MsBackendRawFileReader` is fully compliant with the **Spectra** class as defined by the RforMS initiative and allows highly efficient and straightforward data mining directly in R.

1 Test data

The Bioconductor **tartare** package [2] provides raw files (spectra) recorded on recent Thermo Scientific mass spectrometers.

```
R> library(tartare)
R> eh <- ExperimentHub()
R> query(eh, c('tartare'))


ExperimentHub with 4 records
# snapshotDate(): 2019-10-22
# $dataprovider: Functional Genomics Center Zurich (FGCZ)
# $species: NA
# $rdataclass: Spectra
# additional mcols(): taxonomyid, genome, description, coordinate_1_based, maintainer,
#   rdatadateadded, preparerclass, tags, rdatapath, sourceurl, sourcetype
# retrieve records with, e.g., 'object[["EH3219"]]'

           title
  EH3219 | Q Exactive HF-X mzXML
  EH3220 | Q Exactive HF-X raw
  EH3221 | Fusion Lumos mzXML
  EH3222 | Fusion Lumos raw


R> files <- getFilename(eh)
```

2 Implementation

The **rawR** codebase operates on multiple language and hierarchy levels. High level R functions call `C#` definitions to directly access mass spectrometry data stored in vendor-formatted binary files (*.raw). These levels are connected by an in-process bridge (`rDotNet`) for the .NET framework (see Figure 1). This section gives an overview of the **rawR** `C#` definitions and their execution from the R command line.

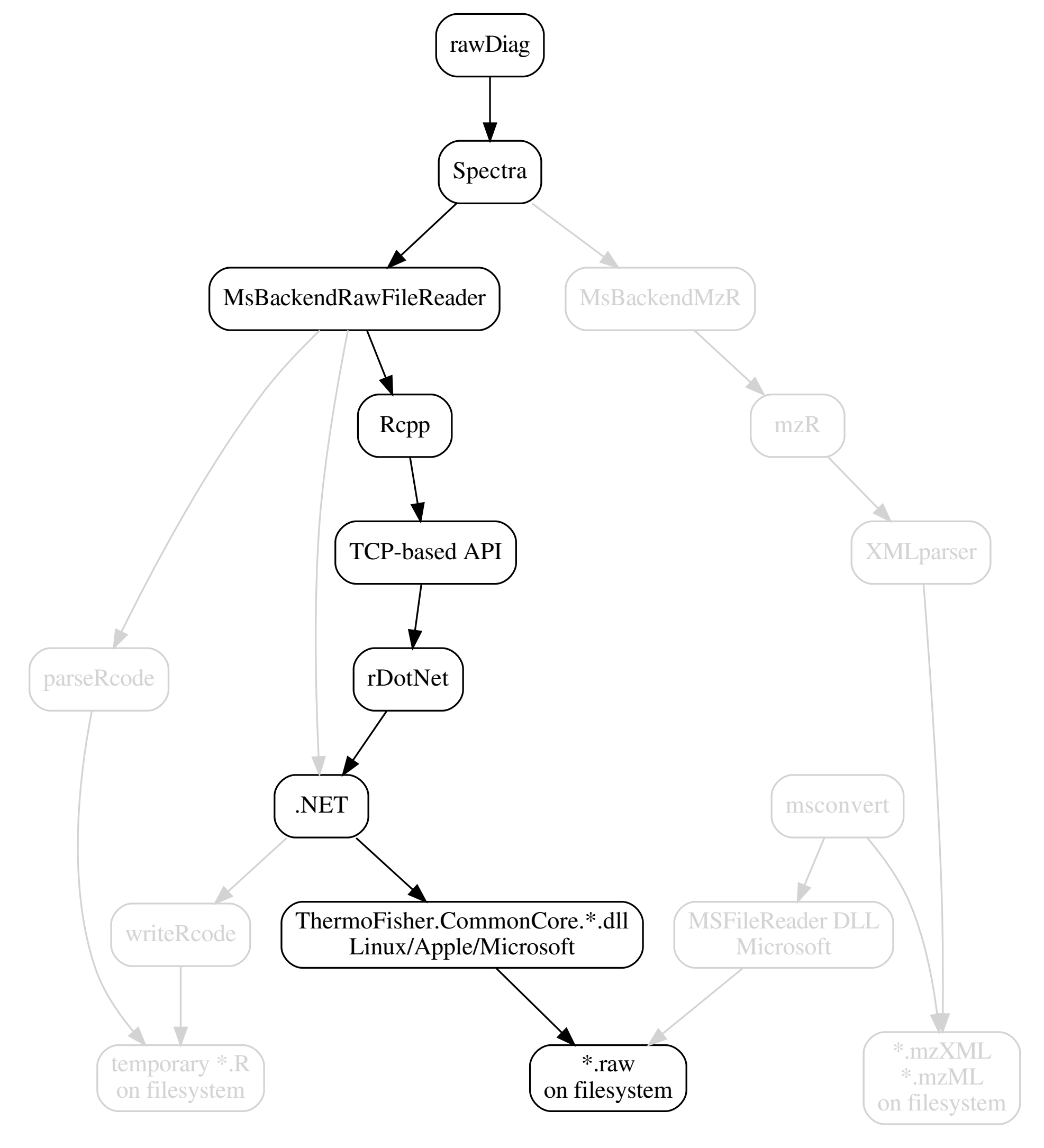

Figure 1: The graph depicts the embedding into the RforMS packages. The node .NET is explained in section 2.1. The proof-of-concept of the MsBackendRawfileReader is drafted in section 2.2. The ".NET libraries are loaded by a runner CLRServer.exe that provides a TCP-based API, giving full visibility into your library(ies)." [3, README]. An application on using the **Spectra** class with **rawDiag** [4] is demonstrated at the end.

2.1 Calling the vendor library from `C#`

The following listing demonstrates the interaction with the vendor specific libraries (ThermoFisher.ComonCore*.dll) in `C#`. The vendor libraries provide access to the raw file structure.

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Runtime.ExceptionServices;
using System.Collections;
using System.Linq;
using ThermoFisher.CommonCore.Data;
using ThermoFisher.CommonCore.Data.Business;
using ThermoFisher.CommonCore.Data.FilterEnums;
using ThermoFisher.CommonCore.Data.Interfaces;
using ThermoFisher.CommonCore.MassPrecisionEstimator;
using ThermoFisher.CommonCore.RawFileReader;


namespace MsBackendRawFileReader


    public class Rawfile


        private IRawDataPlus rawFile;

        public Rawfile(string rawfile)
```

At the moment there are 40 "getter" methods implemented. Below we list all method signatures used on this poster:

```
        public bool IsValidFilter(string filter)

        public string[] GetAutoFilters()

        public double GetRTinSeconds(int scanNumber) {

        public double[] GetSpectrumNoises(int scanNumber, string scanFilter)

        public double[] GetSpectrumCharges(int scanNumber, string scanFilter)

        public double[] GetSpectrumResolutions(int scanNumber, string scanFilter)

        public double[] GetSpectrumIntensities(int scanNumber, string scanFilter)

        public double[] GetSpectrumMasses(int scanNumber)
```

2.2 Calling the `C#` methods from R

The snippet below calls the constructor for a given rawfile.

```
R> library(MsBackendRawFileReader)
R> ##  https://CRAN.R-project.org/package=rDotNet
R> ## create an object and call a method
R> rawfile <- files[4]
R> (x <- rDotNet::.cnew ("Rawfile", rawfile))


<dotnet obj: 4, class: MsBackendRawFileReader.Rawfile, value: "MsBackendRawFileReader.Rawfile">
```

An implementation in `C#` of extracting the mass over charge (m/z) values of a spectrum is listed below.

```
        public double[] GetSpectrumMasses(int scanNumber)
    {
      var scanStatistics = rawFile.GetScanStatsForScanNumber(scanNumber);
      var centroidStream = rawFile.GetCentroidStream(scanNumber, false);

      if (scanStatistics.IsCentroidScan && centroidStream.Length > 0)
      {
        return centroidStream.Masses.ToArray();
      }
      else
      {
        var segmentedScan = rawFile.GetSegmentedScanFromScanNumber(scanNumber, scanStatistics);
        return segmentedScan.Positions.ToArray();
      }
    }
```

Once the .NET library is registered their methods can be called from R as follows: "The R api uses the $ syntax to reference members much like other R object approaches." [3, README]. Here is a code snippet in R extracting peak attributes from a single spectrum.

```
R> # Define scan number 4034
R> scan <- 4034
R> ## getting a vector of mass values
R> head(mZ <- x$GetSpectrumMasses(scan))


[1] 110.0709 111.0744 114.2004 120.0805 129.1020 138.0659


R> ## getting a vector of intensities
R> head(intensity <- x$GetSpectrumIntensities(scan))


[1] 328132.188  11543.382   4682.475  20583.205   7574.656   6460.671


R> ## getting data not available from mzXML
R> noise <- x$GetSpectrumNoises(scan)
R> resolution <- x$GetSpectrumResolutions(scan)
R> charge <- x$GetSpectrumCharges(scan)


R> # https://CRAN.R-project.org/package=protViz
R> pp <- protViz::peakplot("HTFSGVASVESSSGEAFHVGK", list(mZ = mZ, intensity = intensity),
    FUN = function(b,y){cbind(b,y)}, itol = 0.01, col = 'lightgrey')
R> legend("top", "", title=x$GetScanFilter(scan), cex=1.5, bty = "n")
R> lines(mZ, noise, col='#FF111199', lwd=2)
R> points(mZ,intensity, pch=16, col = as.factor(charge), cex=0.5)
R> legend("topleft", paste(unique(charge), "+"), pch=16, col=as.factor(unique(charge)))
```

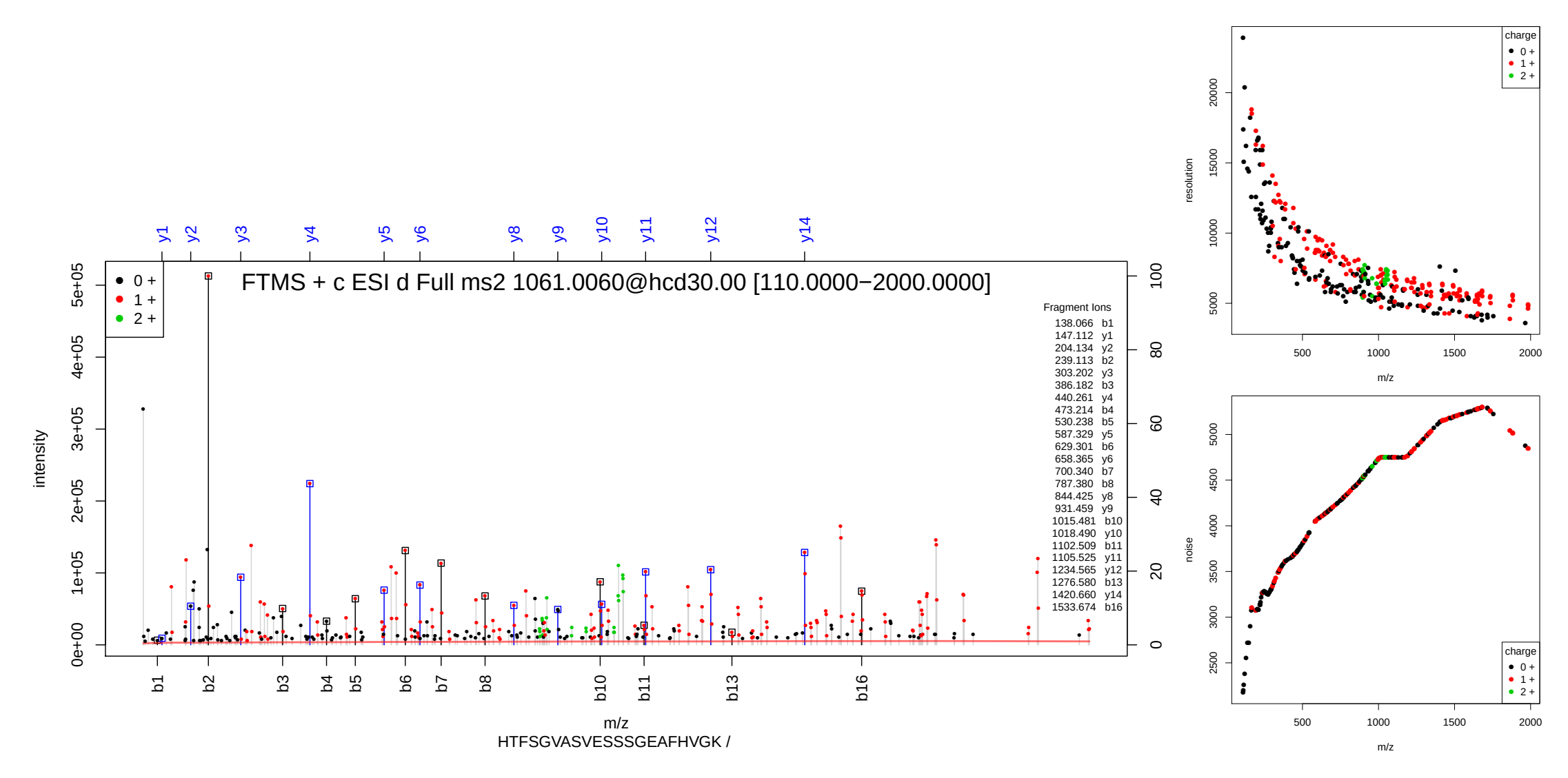

Figure 2: The graphic depicts the peptide spectrum match (PSM) with the `"HTFSGVASVESSSGEAFHVGK"` peptide using `https://CRAN.R-project.org/package=protViz` of scan 4034. The peptide assignment was determined by precursor mass and a scoring function based on counting fragment ion matches. The graphics on the bottom plot the resolution and noise characteristics as a function of mass over charge.

Scan collections can be subset using native filters on the vendor library level, see also [1, page 28].

```
R> head(x$GetAutoFilters())


[1] "FTMS + c ESI Full ms [350.0000-1500.0000]"
[2] "FTMS + c ESI Full ms [350.0000-1800.0000]"
[3] "FTMS + c ESI Full ms [350.0000-2000.0000]"
[4] "FTMS + c ESI Full ms [375.0000-1500.0000]"
[5] "FTMS + c ESI d Full ms2 376.6913@hcd30.00 [110.0000-764.0000]"
[6] "FTMS + c ESI d Full ms2 378.2359@hcd30.00 [110.0000-767.0000]"


R> x$IsValidFilter("most likely not a valid filter string")


[1] FALSE


R> filterString <- "FTMS + c ESI Full ms [375-1500]"
R> x$IsValidFilter(filterString)


[1] TRUE
```

A direct application of filters is the generation of extracted ion chromatograms (XIC). Also here we use the proposed vendor implementation.

```
R> # determine the peptide weight
R> (mZ <- (protViz::parentIonMass("HTFSGVASVESSSGEAFHVGK") + 1.008) / 2)


[1] 1060.506


R> X <- x$GetXIC(mZ, tol = 5, filter = filterString)
```

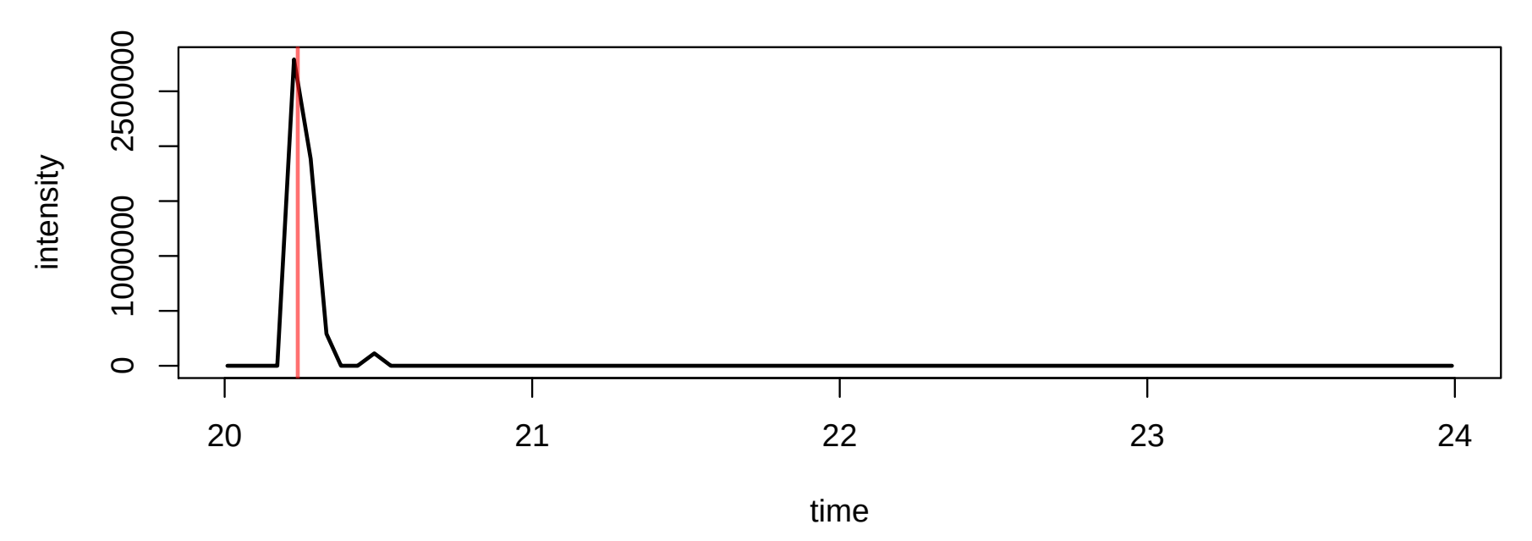

Figure 3: XIC of mass 1060.5061 Da within a mass window of 5 ppm applying the filter FTMS + c ESI Full ms [375-1500]. The red linie marks the retention time of 20.2 seconds when the MS2 (scan 4034) shown in Figure 2 was taken.

3 Usage

We decided to use the object-oriented representation of spectral data as defined by the RforMS initiative in the **Spectra** package. The core idea is to *untangle* functionality and data origin/storage (see `https://github.com/jorainer/swemsa_2019`). To demonstrate the power of this concept we cross compare data access through `MsBackendMzR` with `MsBackendRawFileReader`.

```
R> BiocManager::install('cpanse/MsBackendRawFileReader')



R> # initialize bioconductor tartare package data
R> be <- lapply(files, function(f){
    if (grepl("mzXML$", f))
      backendInitialize(Spectra::MsBackendMzR(), files = f)
    else
      backendInitialize(MsBackendRawFileReader(), files = f, extra=TRUE)
  })
R> S <- lapply(be, Spectra)
R> # Howto merge different backends?
R> sapply(S, class)



   EH3219    EH3220    EH3221    EH3222
"Spectra" "Spectra" "Spectra" "Spectra"
```

The following lines plot the scan frequency on all recorded MS level of all four files using **rawDiag** [4] plot methods on the **Spectra** objects bound to different backends.

```
R> library(rawDiag)
R> gpl <- lapply(S, function(x){ rawDiag::Plot(x, FUN = 'ScanFrequency')})
```

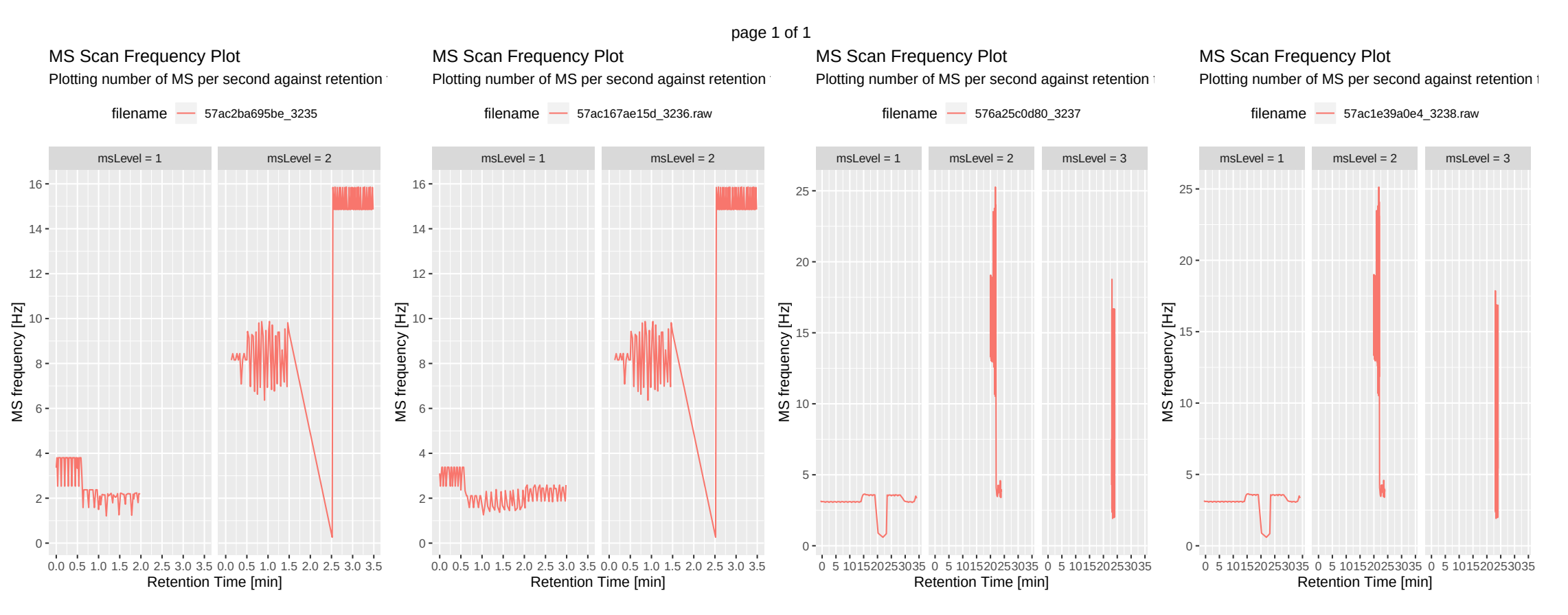

Figure 4: Coplots compare scan frequencies extracted from two raw files accessed by two backend implementations (2x2 design). Left-to-right: Q Exactive mzXML and raw; Fusion Lumos mzXML, raw.

4 Discussion and outlook

As shown in Figure 5 the `MsBackendMzR` exposes only a small number of scan metadata and reads those into memory during the backend initialization process.

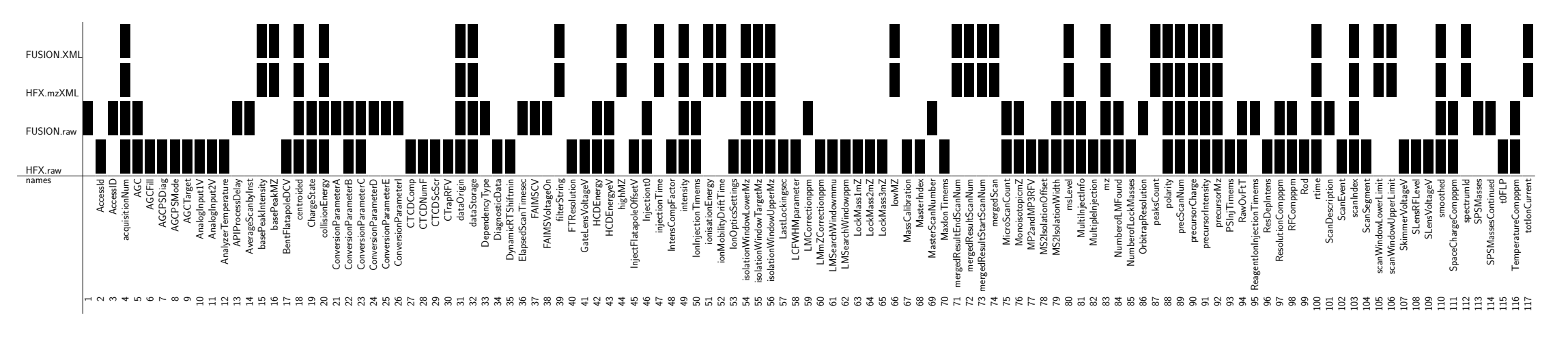

Figure 5: The table displays the scan meta data of two Orbitrap instruments using raw and mzXML files. The availablilty of a variable is inidicated by a black rectangle.

In contrast, the `MsBackendRawFileReader` reads only a few "run-level" metadata, e.g., instrument id or file name during the initialization. The different philosophies in the backend implementations are also visible in the benchmark below.

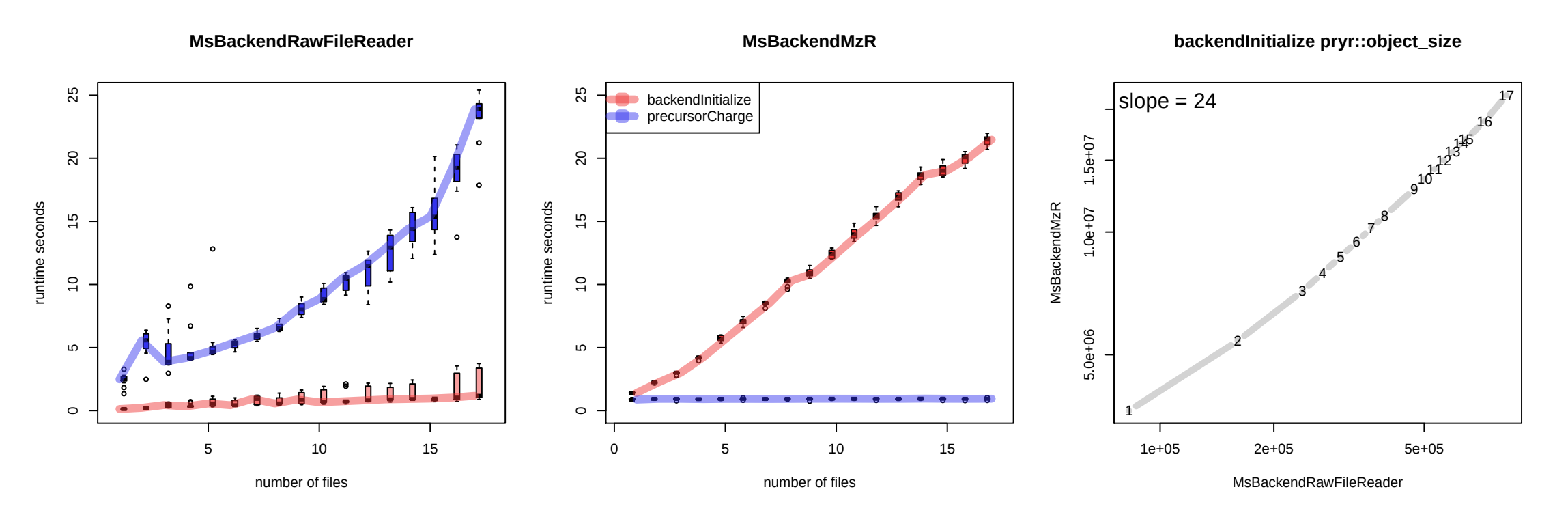

Figure 6: The boxplots show ten repetitions of *initializing* and *extracting the precursor charge information* of a batch of `1:17` raw and mzXML files on a single-core computer. The scatter plot on the right compares the memory size in Bytes of the objects returned by both `backendInitialize` function calls.

In the near future we hope to fully synchronize **rawR** with low-level infrastructure packages like **Spectra** and **Chromatograms**. Once a stable interoperate ability is achieved, it will be submitted to Bioconductor in order to disseminate **rawR** into the mass spectrometry community.

References

[1] Jim Shofstahl. Reading Files using the C# file reader, December 2015. URL: `http://planetorbitrap.com/rawfilereader#.WjkqIUtJmL4`.

[2] Christian Panse and Tobias Kockmann. *tartare: Raw ground spectra recorded on Thermo Fisher Scientific mass spectrometers*, 2019. R package version 1.0.0. URL: `https://bioconductor.org/packages/tartare/`.

[3] Jonathan Shore. *rDotNet: Low-Level Interface to the '.NET' Virtual Machine Along the Lines of the R C/Call API*, 2017. R package version 0.9.1. URL: `https://CRAN.R-project.org/package=rDotNet`.

[4] Christian Trachsel, Christian Panse, Tobias Kockmann, Witold E. Wolski, Jonas Grossmann, and Ralph Schlapbach. rawDiag: An R Package Supporting Rational LC–MS Method Optimization for Bottom-up Proteomics. *Journal of Proteome Research*, 17(8):2908–2914, July 2018. URL: `https://doi.org/10.1021/acs.jproteome.8b00173`, `doi:10.1021/acs.jproteome.8b00173`.

Contact: Functional Genomics Center Zurich, Swiss Federal Institute of Technology Zurich | University of Zurich, Winterthurerstr. 190, CH-8057 Zurich, SWITZERLAND. Phone: +41 44 635 39 12, Fax: +41 44 635 39 22, EMail: `cp@fgcz.ethz.ch` | `Tobias.Kockmann@fgcz.ethz.ch`, URL: `http://fgcz.ethz.ch`.