Package 'readJDX'

May 9, 2026

Type Package

Title Import Data in the JCAMP-DX Format

Version 0.6.4

Date 2023-11-18

Description Import data written in the JCAMP-DX format. This is an instrument-independent format used in the field of spectroscopy. Examples include IR, NMR, and Raman spectroscopy. See the vignette for background and supported formats. The official JCAMP-DX site is <http://www.jcamp-dx.org/>.

URL https://github.com/bryanhanson/readJDX

BugReports https://github.com/bryanhanson/readJDX/issues

License GPL (>= 3)

ByteCompile TRUE

Depends R (>= 3.0)

Suggests knitr, rmarkdown, bookdown, mvbutils

Imports stringr

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation no

Author Bryan A. Hanson [aut, cre] (ORCID: <https://orcid.org/0000-0003-3536-8246>)

Maintainer Bryan A. Hanson <hanson@depauw.edu>

Repository CRAN

Date/Publication 2023-11-18 18:20:02 UTC

Contents

`decompLines` *Decompress the Data Lines in a JCAMP-DX File*

Description

This function is used by the package, and can also be called by the user who wishes to study problematic lines in a file in a standalone manner. See the examples.

Usage

```
decompLines(VL, debug = 0)
```

Arguments

`VL`	Character. The variable list to be processed as a character vector. Depending upon the `mode` and the stage of processing: • the lines may be named as "Line_1" etc • a format code may be present as the first entry • there may be other non-numeric lines that eventually get stripped off
`debug`	Integer. See `readJDX` for details.

Value

A list of numeric strings, the result of unpacking the compressed data. One list element is returned for each line. The numeric string is named with the ASDF compression mode.

Details

Each individual line passed here is converted to a list to make processing and naming the pieces easier.The individual entries are named according to their *original* ASDF compression code. Unless otherwise noted all functions called from here should act on a list of lines, via lapply. Note too that the X values are present, e.g. X, Y1, Y2 ... Yn.

Formats

AFFN is separated by any amount of white space so processing is straightforward, as exponents are handled automatically, and white space stripped off automatically, courtesy of R internals. It appears AFFN is never mixed with other formats. The other formats are collectively called ASDF in the standard.

Examples

```
testLines1 <- c(
  "482A885145L989378k853295J46714q39581j382088R41076k774051J365135l135709P53917",
  "472a903359j71857q18832K573831k615133L481852l395846L8948441478693M916433",
  "463B483240m513444O146172m826168N233079m522551M000252m097028L4661111460183",
  "454i0520L061628k524598K788931k509430L219286k511160K709095k122775J594246"
)
demo1 <- decompLines(testLines1, debug = 6)

testLines2 <- c(
  # EU AFFN:
  "1898,58486802541 -0,0170190036296844 -0,0170874372124672 -0,0171865783631802",
  "1917,23501403401 -0,0176097713410854 -0,0177919361740351 -0,0179808251559734",
  # AFFN with fixed field width/extra space:
  "           16383       2259260      -5242968      -7176216      -1616072",
  "           16379      10650432       4373926      -3660824       2136488"
)

demo2 <- decompLines(testLines2, debug = 6)
```

`readJDX` *Import a File Written in the JCAMP-DX Format*

Description

This function supervises the entire import process. Not all official formats are supported, see the vignettes. Prior to release, this package is checked against a very large number of files in the author's collection. However, the JCAMP-DX standard allows many variations and it is difficult to anticipate all permutations. Error messages will generally let you know what's going on. If you have a file that you feel should be supported but gives an error, please file an issue at Github and share the file.

Usage

```
readJDX(file = "", SOFC = TRUE, debug = 0)
```

Arguments

`file`	Character. The file name to import.
`SOFC`	Logical. "Stop on Failed Check" The default is `TRUE` i.e. stop when something is not right. This ensures that correct data is returned. Change to `FALSE` at your own risk. NOTE: Only certain checks can be skipped via this option, as there are some parameters that must be available and correct in order to return *any* answer. For instance, one must end up with the same number of X and Y values. This option is provided for those **advanced users** who have carefully checked their original files and want to skip the required checks. It may also be useful for troubleshooting. The JCAMP-DX standard typically requires several checks of the data as it is decompressed. These checks are essential to obtaining the

correct results. However, some JCAMP-DX writing programs do not follow the standard to the letter. For instance we have observed that not all JCAMP-DX writers put FIRSTY into the metadata, even though it is required by the standard. In other cases values in the file have low precision (see section on precision). Another example we have observed is NMR files where the X values are the count/index of data points, and FIRSTY is given in Hz. Since the field strength and center of the sweep frequency are needed to convert to ppm, and these are parameters not required in the standard, one cannot return an answer in either ppm or Hz automatically. In cases like this, one can set `SOFC = FALSE` and then manually convert the X axis.

`debug` Integer. The level of debug reporting desired. For those options giving a lot of output, you may wish to consider directing the output via `sinkall` and then search the results for the problematic lines.

- 1 or higher = import progress is reported.
- 2 = details about the variable lists, compression formats and parameters that were found.
- 3 = print the extracted X values (huge output!).
- 4 = detailed info on the Y value processing (huge output!).
- 5 = detailed info about processing the Y values when DUP is in use (huge output!).
- 6 = detailed info about processing the Y values when DIF is in use (huge output!).

In cases when an error is about to stop execution, you get additional information regardless of the `debug` value.

Value

A list, as follows:

- The first element is a data frame summarizing the pieces of the imported file.
- The second element is the file metadata.
- The third element is a integer vector giving the comment lines found (exclusive of the metdata, which typically contains many comments).

Additional elements contain the extracted data as follows:

- If the file contains one non-NMR spectrum, or a processed NMR spectrum (i.e. only the final real data), a single data frame.
- If the file contains the real and imaginary parts of a 1D NMR spectrum, there will be two data frames, one containing the real portion and the other the imaginary portion.
- In all cases above, the data frame has elements `x` and `y`.
- In the case of 2D NMR data, additional named list elements are returned including the F2 frequency values, the F1 frequency values, and a matrix containing the 2D data.
- In the case of LC-MS or GC-MS data, a data frame is returned for each time point. The data frame has elements `mz` and `int` (intensity). Each time point is named with the time from the file.

Included Data Files

The examples make use of data files included with the package:

- File `SBO.jdx` is an IR spectrum of Smart Balance Original spread (a butter substitute). The spectrum is presented in transmission format, and was recorded on a ThermoFisher instrument. The file uses AFFN compression, and was written with the JCAMP-DX 5.01 standard. Note that even though the Y-axis was recorded in percent transmission, in the JDX file it is stored on [0...1].
- File `PCRF.jdx` is a 1H NMR spectrum of a hexane extract of a reduced fat potato chip. The spectrum was recorded on a JEOL instrument. The file uses SQZ DIF DUP compression, and was written with the JCAMP-DX 6.00 standard.
- File `PCRF_line265.jdx` has a deliberate error in it.
- File `isasspc1.jdx` is a 2D NMR file recorded on a JEOL GX 400 instrument. The file is freely available at http://www.jcamp-dx.org/.
- File `MiniDIFDUP.JDX` is a small demonstration file, used in the vignettes to illustrate the decompression process. It is derived from a freely available file.

Precision

Internally, this package uses a tolerance factor when comparing values during certain checks. This is desirable because the original values in the files are text strings of varying lengths which get converted to numerical values by `R`. Occasionally values in the file, such as FIRSTY, are stored with low precision, and the computation of the value to be compared occurs with much greater precision. In these cases the check can fail even when the tolerance is pretty loose. In these cases one might consider setting `SOFC = FALSE` to allow the calculation to proceed. If you do this, be certain to check the results carefully as described under `SOFC`.

Y Value Check

The standard requires a "Y Value Check" when in DIF mode. Extra Y values have been appended to each line to use in the check, and the last Y value on a line must equal the first Y value on the next line *IFF* one is in DIF mode. After a successful check, the extra Y value must be removed. In actual practice, some vendors, at least some of the time, seem to differ as to the meaning of "being in DIF mode". In turn, this determines how the Y value check should proceed.

- The standard says "When, and only when, the last ordinate of a line is in DIF form ... The first ordinate of the next line ... is always an actual value, equal to the last calculated ordinate of the previous line". See section 5.8.3 of the 1988 publication.
- Taking this definition literally, the Y value check (and removal of the extra value), should occur when one sees e.g. ... DIF DIF DIF (end of line). Let's call this "literal DIF". A literal DIF is easy to detect and act on.
- In other cases, something like ... DIF DUP DUP (end of line) is considered to be in DIF mode for Y value check purposes. In these cases we have look backwards to see if we are in DIF mode. Let's call this "relayed DIF".
- However, some vendors may treat ... DIF DUP DUP (end of line) as not in DIF mode, and hence one should not do the Y value check and not remove any values, as this vendor would not have added an extra Y value.

- In addition to these three possibilities, readJDX through versions 0.3.xx used a different definition, namely if there were any DIF entries anywhere on the line, then DIF mode was assumed and the Y value check carried out. This worked for many files, but not all.
- In the 0.4.xx series, readJDX detects both the literal and relayed definitions and tries to keep moving forward as much as possible.

Performance

readJDX is not particularly fast. Priority has been given to assuring correct answers, helpful debugging messages and understandable code.

See Also

Do browseVignettes("readJDX") for background information, references, supported formats, and details about the roles of each function. If you have a multiblock file (which contains multiple spectra, but not 2D NMR, LC-MS or GC-MS data sets), please see splitMultiblockDX for a function to break such files into individual files which can then be processed in the normal way.

Examples

```
# IR spectrum
sbo <- system.file("extdata", "SBO.jdx", package = "readJDX")
chk <- readJDX(sbo)
plot(chk[[4]]$x, chk[[4]]$y / 100,
  type = "l", main = "Original Smart Balance Spread",
  xlab = "wavenumber", ylab = "Percent Transmission"
)

# 1H NMR spectrum
pcrf <- system.file("extdata", "PCRF.jdx", package = "readJDX")
chk <- readJDX(pcrf)
plot(chk[[4]]$x, chk[[4]]$y,
  type = "l", main = "Reduced Fat Potato Chip Extract",
  xlab = "ppm", ylab = "Intensity"
)

# Capturing processing for troubleshooting
mdd <- system.file("extdata", "MiniDIFDUP.JDX", package = "readJDX")
tf <- tempfile(pattern = "Troubleshooting", fileext = "txt")
sinkall(tf)
chk <- readJDX(mdd, debug = 6)
sinkall() # close the file connection
file.show(tf)

# 2D HETCORR spectrum
## Not run:
nmr2d <- system.file("extdata", "isasspc1.dx", package = "readJDX")
chk <- readJDX(nmr2d)
contour(chk$Matrix, drawlabels = FALSE) # default contours not optimal

## End(Not run)
```

```
## Not run:
# Line 265 has an N -> G error.  Try with various levels of debug.
# Even with debug = 0 you get useful diagnostic info.
problem <- system.file("extdata", "PCRF_line265.jdx", package = "readJDX")
chk <- readJDX(problem)

## End(Not run)
```

`sinkall` *Divert Both stdout and stderr to a File*

Description

This is a simple utility function to direct the output of stdout and stderr to a file. stdout is the information normally printed in the console, for instance the results of `print(rnorm(5))`. stderr is the output created by functions `message`, `warning` and `stop`. The purpose of this function is to allow one to direct all this output into a single file where the results can be studied, for instance, for troubleshooting purposes. Works exactly like the base `sink()` function: you have to call it a second time with no arguments to close the file.

Usage

```
sinkall(filename = NULL)
```

Arguments

`filename`	Character. A path to a filename where the results will be captured.

Value

`NULL`, invisibly.

Examples

```
## Not run:
tf <- tempfile(pattern = "SinkDemo", fileext = "txt")
sinkall(tf)
print("Hello")
print(rnorm(2))
print(normr(2)) # typo, so it errors
message("A message from message()")
warning("A warning from warning()")
cat("Information via cat\\n")
sinkall() # one must close the file connection
file.show(tf)

## End(Not run)
```

splitMultiblockDX *Split a Multiblock JCAMP-DX File into Individual Files*

Description

Process a multiblock JCAMP-DX file into separate files, one for each data set.

Usage

```
splitMultiblockDX(file = NULL, subdir = "data_sets")
```

Arguments

file	Length one character giving the name of the multiblock file.
subdir	Character. If not NULL, a subdirectory by this name will be created in the working directory, and the individual files will be placed in this subdirectory. If the subdirectory already exists, an error will be thrown.

Value

A list is returned invisibly that has one DX data set per list element. **Individual files are written into the current directory if** subdir = NULL**, so you probably want to work on a copy of the data in this case**. File names are extracted from the ##TITLE= sample_name field and will be sample_name (more precisely, the name will be whatever follows ##TITLE= on the line). If sample names are duplicated the output files will be overwritten as the file is processed (with a warning). In this case you should open the multiblock file in a plain text editor and search for ##TITLE=, then edit the names to be unique before running this function. Also, you should ensure that sample_name will result in a valid file name on your operating system.

Warning

Individual data sets are written into the current directory if you set subdir **to** NULL**, so you may want to work on a copy of the data in this case**.

Author(s)

Bryan A. Hanson, DePauw University.

Index