Package 'readtext'

October 14, 2022

Version 0.81

Type Package

Title Import and Handling for Plain and Formatted Text Files

Description Functions for importing and handling text files and formatted text files with additional meta-data, such including '.csv', '.tab', '.json', '.xml', '.html', '.pdf', '.doc', '.docx', '.rtf', '.xls', '.xlsx', and others.

License GPL-3

Depends R (>= 3.6)

Imports antiword, data.table, digest, httr, jsonlite (>= 0.9.10), pdftools, readODS (>= 1.7.0), readxl, streamR, stringi, striprtf, tibble, xml2, utils

Suggests knitr, pkgload, rmarkdown, quanteda (>= 3.0), testthat

URL https://github.com/quanteda/readtext

Encoding UTF-8

BugReports https://github.com/quanteda/readtext/issues

LazyData TRUE

VignetteBuilder knitr

RoxygenNote 7.1.1

NeedsCompilation no

Author Kenneth Benoit [aut, cre, cph],
Adam Obeng [aut],
Kohei Watanabe [ctb],
Akitaka Matsuo [ctb],
Paul Nulty [ctb],
Stefan Müller [ctb]

Maintainer Kenneth Benoit <kbenoit@lse.ac.uk>

Repository CRAN

Date/Publication 2021-07-14 14:40:02 UTC

R topics documented:

`readtext-package` *Import and handling for plain and formatted text files*

Description

A set of functions for importing and handling text files and formatted text files with additional meta-data, such including .csv, .tab, .json, .xml, .xls, .xlsx, and others.

Details

readtext makes it easy to import text files in various formats, including using operating system filemasks to load in groups of files based on glob pattern matches, including files in multiple directories or sub-directories. **readtext** can also read multiple files into R from compressed archive files such as .gz, .zip, .tar.gz, etc. Finally **readtext** reads in the document-level meta-data associated with texts, if those texts are in a format (e.g. .csv, .json) that includes additional, non-textual data.

Package options

`readtext_verbosity` Default verbosity for messages produced when reading files. See `readtext()`.

Author(s)

Ken Benoit, Adam Obeng, and Paul Nulty

`as.character.readtext` *return only the texts from a readtext object*

Description

An accessor function to return the texts from a readtext object as a character vector, with names matching the document names.

Usage

```
## S3 method for class 'readtext'
as.character(x, ...)
```

Arguments

x	the readtext object whose texts will be extracted
...	further arguments passed to or from other methods

---

data_char_encodedtexts

encoded texts for testing

Description

data_char_encodedtexts is a 10-element character vector with 10 different encodings

Usage

```
data_char_encodedtexts
```

Format

An object of class character of length 10.

Examples

```
## Not run:
Encoding(data_char_encodedtexts)
data.frame(labelled = names(data_char_encodedtexts),
           detected = encoding(data_char_encodedtexts)$all)

## End(Not run)
```

---

data_files_encodedtexts

a .zip file of texts containing a variety of differently encoded texts

---

Description

A set of translations of the Universal Declaration of Human Rights, plus one or two other miscellaneous texts, for testing the text input functions that need to translate different input encodings.

Source

The Universal Declaration of Human Rights resources, http://www.ohchr.org/EN/UDHR/Pages/SearchByLang.aspx

Examples

```
## Not run: # unzip the files to a temporary directory
FILEDIR <- tempdir()
unzip(system.file("extdata", "data_files_encodedtexts.zip", package = "readtext"),
      exdir = FILEDIR)

# get encoding from filename
filenames <- list.files(FILEDIR, "\\.txt$")
# strip the extension
filenames <- gsub(".txt$", "", filenames)
parts <- strsplit(filenames, "_")
fileencodings <- sapply(parts, "[", 3)
fileencodings

# find out which conversions are unavailable (through iconv())
cat("Encoding conversions not available for this platform:")
notAvailableIndex <- which(!(fileencodings %in% iconvlist()))
fileencodings[notAvailableIndex]

# try readtext
require(quanteda)
txts <- readtext(paste0(FILEDIR, "/", "*.txt"))
substring(texts(txts)[1], 1, 80) # gibberish
substring(texts(txts)[4], 1, 80) # hex
substring(texts(txts)[40], 1, 80) # hex

# read them in again
txts <- readtext(paste0(FILEDIR,  "/", "*.txt"), encoding = fileencodings)
substring(texts(txts)[1], 1, 80)  # English
substring(texts(txts)[4], 1, 80)  # Arabic, looking good
substring(texts(txts)[40], 1, 80) # Cyrillic, looking good
substring(texts(txts)[7], 1, 80)  # Chinese, looking good
substring(texts(txts)[26], 1, 80) # Hindi, looking good

txts <- readtext(paste0(FILEDIR, "/", "*.txt"), encoding = fileencodings,
                 docvarsfrom = "filenames",
                 docvarnames = c("document", "language", "inputEncoding"))
encodingCorpus <- corpus(txts, source = "Created by encoding-tests.R")
summary(encodingCorpus)

## End(Not run)
```

`encoding` *detect the encoding of texts*

Description

Detect the encoding of texts in a character readtext object and report on the most likely encoding for each document. Useful in detecting the encoding of input texts, so that a source encoding can be (re)specified when inputting a set of texts using `readtext()`, prior to constructing a corpus.

Usage

```
encoding(x, verbose = TRUE, ...)
```

Arguments

x	character vector, corpus, or readtext object whose texts' encodings will be detected.
verbose	if FALSE, do not print diagnostic report
...	additional arguments passed to stri_enc_detect

Details

Based on stri_enc_detect, which is in turn based on the ICU libraries. See the ICU User Guide, http://userguide.icu-project.org/conversion/detection.

Examples

```
## Not run: encoding(data_char_encodedtexts)
# show detected value for each text, versus known encoding
data.frame(labelled = names(data_char_encodedtexts),
           detected = encoding(data_char_encodedtexts)$all)

# Russian text, Windows-1251
myreadtext <- readtext("https://kenbenoit.net/files/01_er_5.txt")
encoding(myreadtext)

## End(Not run)
```

---

readtext *read a text file(s)*

Description

Read texts and (if any) associated document-level meta-data from one or more source files. The text source files come from the textual component of the files, and the document-level metadata ("docvars") come from either the file contents or filenames.

Usage

```
readtext(
  file,
  ignore_missing_files = FALSE,
  text_field = NULL,
  docid_field = NULL,
  docvarsfrom = c("metadata", "filenames", "filepaths"),
  dvsep = "_",
  docvarnames = NULL,
```

```
  encoding = NULL,
  source = NULL,
  cache = TRUE,
  verbosity = readtext_options("verbosity"),
  ...
)
```

Arguments

`file` the complete filename(s) to be read. This is designed to automagically handle a number of common scenarios, so the value can be a "glob"-type wildcard value. Currently available filetypes are:

Single file formats:

`txt` plain text files: So-called structured text files, which describe both texts and metadata: For all structured text filetypes, the column, field, or node which contains the the text must be specified with the `text_field` parameter, and all other fields are treated as docvars.

`json` data in some form of JavaScript Object Notation, consisting of the texts and optionally additional docvars. The supported formats are:

- a single JSON object per file
- line-delimited JSON, with one object per line
- line-delimited JSON, of the format produced from a Twitter stream. This type of file has special handling which simplifies the Twitter format into docvars. The correct format for each JSON file is automatically detected.

`csv,tab,tsv` comma- or tab-separated values

`html` HTML documents, including specialized formats from known sources, such as Nexis-formatted HTML. See the `source` parameter below.

`xml` XML documents are supported – those of the kind that can be read by `xml2::read_xml()` and navigated through `xml2::xml_find_all()`. For xml files, an additional argument `collapse` may be passed through `...` that names the character(s) to use in appending different text elements together.

`pdf` pdf formatted files, converted through **pdftools**.

`odt` Open Document Text formatted files.

`doc, docx` Microsoft Word formatted files.

`rtf` Rich Text Files.

Reading multiple files and file types:

In addition, `file` can also not be a path to a single local file, but also combinations of any of the above types, such as:

a wildcard value any valid pathname with a wildcard ("glob") expression that can be expanded by the operating system. This may consist of multiple file types.

a URL to a remote which is downloaded then loaded

`zip,tar,tar.gz,tar.bz` archive file, which is unzipped. The contained files must be either at the top level or in a single directory. Archives, remote URLs and glob patterns can resolve to any of the other filetypes, so you

could have, for example, a remote URL to a zip file which contained Twitter JSON files.

`ignore_missing_files`
: if `FALSE`, then if the file argument doesn't resolve to an existing file, then an error will be thrown. Note that this can happen in a number of ways, including passing a path to a file that does not exist, to an empty archive file, or to a glob pattern that matches no files.

`text_field, docid_field`
: a variable (column) name or column number indicating where to find the texts that form the documents for the corpus and their identifiers. This must be specified for file types `.csv`, `.json`, and `.xls/.xlsx` files. For XML files, an XPath expression can be specified.

`docvarsfrom`
: used to specify that docvars should be taken from the filenames, when the `readtext` inputs are filenames and the elements of the filenames are document variables, separated by a delimiter (`dvsep`). This allows easy assignment of docvars from filenames such as `1789-Washington.txt`, `1793-Washington`, etc. by `dvsep` or from meta-data embedded in the text file header (`headers`). If `docvarsfrom` is set to `"filepaths"`, consider the full path to the file, not just the filename.

`dvsep`
: separator (a regular expression character string) used in filenames to delimit docvar elements if `docvarsfrom="filenames"` or `docvarsfrom="filepaths"` is used

`docvarnames`
: character vector of variable names for `docvars`, if `docvarsfrom` is specified. If this argument is not used, default docvar names will be used (`docvar1`, `docvar2`, ...).

`encoding`
: vector: either the encoding of all files, or one encoding for each files

`source`
: used to specify specific formats of some input file types, such as JSON or HTML. Currently supported types are `"twitter"` for JSON and `"nexis"` for HTML.

`cache`
: if `TRUE`, save remote file to a temporary folder. Only used when `file` is a URL.

`verbosity`
: - 0: output errors only
- 1: output errors and warnings (default)
- 2: output a brief summary message
- 3: output detailed file-related messages

`...`
: additional arguments passed through to low-level file reading function, such as `file()`, `fread()`, etc. Useful for specifying an input encoding option, which is specified in the same was as it would be give to `iconv()`. See the Encoding section of file for details.

Value

a data.frame consisting of a columns `doc_id` and `text` that contain a document identifier and the texts respectively, with any additional columns consisting of document-level variables either found in the file containing the texts, or created through the `readtext` call.

Examples

```
## Not run:
## get the data directory
if (!interactive()) pkgload::load_all()
DATA_DIR <- system.file("extdata/", package = "readtext")

## read in some text data
# all UDHR files
(rt1 <- readtext(paste0(DATA_DIR, "/txt/UDHR/*")))

# manifestos with docvars from filenames
(rt2 <- readtext(paste0(DATA_DIR, "/txt/EU_manifestos/*.txt"),
                 docvarsfrom = "filenames",
                 docvarnames = c("unit", "context", "year", "language", "party"),
                 encoding = "LATIN1"))

# recurse through subdirectories
(rt3 <- readtext(paste0(DATA_DIR, "/txt/movie_reviews/*"),
                 docvarsfrom = "filepaths", docvarnames = "sentiment"))

## read in csv data
(rt4 <- readtext(paste0(DATA_DIR, "/csv/inaugCorpus.csv")))

## read in tab-separated data
(rt5 <- readtext(paste0(DATA_DIR, "/tsv/dailsample.tsv"), text_field = "speech"))

## read in JSON data
(rt6 <- readtext(paste0(DATA_DIR, "/json/inaugural_sample.json"), text_field = "texts"))

## read in pdf data
# UNHDR
(rt7 <- readtext(paste0(DATA_DIR, "/pdf/UDHR/*.pdf"),
                 docvarsfrom = "filenames",
                 docvarnames = c("document", "language")))
Encoding(rt7$text)

## read in Word data (.doc)
(rt8 <- readtext(paste0(DATA_DIR, "/word/*.doc")))
Encoding(rt8$text)

## read in Word data (.docx)
(rt9 <- readtext(paste0(DATA_DIR, "/word/*.docx")))
Encoding(rt9$text)

## use elements of path and filename as docvars
(rt10 <- readtext(paste0(DATA_DIR, "/pdf/UDHR/*.pdf"),
                  docvarsfrom = "filepaths", dvsep = "[/_.]"))

## End(Not run)
```

readtext_options *Get or set package options for readtext*

Description

Get or set global options affecting functions across **readtext**.

Usage

```
readtext_options(..., reset = FALSE, initialize = FALSE)
```

Arguments

...	options to be set, as key-value pair, same as options(). This may be a list of valid key-value pairs, useful for setting a group of options at once (see examples).
reset	logical; if TRUE, reset all **readtext** options to their default values
initialize	logical; if TRUE, reset only the **readtext** options that are not already defined. Used for setting initial values when some have been defined previously, such as in .Rprofile.

Details

Currently available options are:

verbosity Default verbosity for messages produced when reading files. See readtext().

Value

When called using a key = value pair (where key can be a label or quoted character name)), the option is set and TRUE is returned invisibly.

When called with no arguments, a named list of the package options is returned.

When called with reset = TRUE as an argument, all arguments are options are reset to their default values, and TRUE is returned invisibly.

Examples

```
## Not run:
# save the current options
(opt <- readtext_options())

# set higher verbosity
readtext_options(verbosity = 3)

# read something in here
if (!interactive()) pkgload::load_all()
DATA_DIR <- system.file("extdata/", package = "readtext")
readtext(paste0(DATA_DIR, "/txt/UDHR/*"))
```

```
# reset to saved options
readtext_options(opt)

## End(Not run)
```

Index