Package: epubr (via r-universe)

October 29, 2026

Version 0.6.5

Title Read EPUB File Metadata and Text

Description Provides functions supporting the reading and parsing of internal e-book content from EPUB files. The 'epubr' package provides functions supporting the reading and parsing of internal e-book content from EPUB files. E-book metadata and text content are parsed separately and joined together in a tidy, nested tibble data frame. E-book formatting is not completely standardized across all literature. It can be challenging to curate parsed e-book content across an arbitrary collection of e-books perfectly and in completely general form, to yield a singular, consistently formatted output. Many EPUB files do not even contain all the same pieces of information in their respective metadata. EPUB file parsing functionality in this package is intended for relatively general application to arbitrary EPUB e-books. However, poorly formatted e-books or e-books with highly uncommon formatting may not work with this package. There may even be cases where an EPUB file has DRM or some other property that makes it impossible to read with 'epubr'. Text is read 'as is' for the most part. The only nominal changes are minor substitutions, for example curly quotes changed to straight quotes. Substantive changes are expected to be performed subsequently by the user as part of their text analysis. Additional text cleaning can be performed at the user's discretion, such as with functions from packages like 'tm' or 'qdap'.

License MIT + file LICENSE

Encoding UTF-8

ByteCompile true

URL https://docs.ropensci.org/epubr/, https://github.com/ropensci/epubr

BugReports https://github.com/ropensci/epubr/issues

Suggests testthat, knitr, rmarkdown, readr

Imports xml2, xslt, magrittr, tibble, dplyr, tidyr

VignetteBuilder knitr

RoxygenNote 7.3.2

Config/pak/sysreqs libicu-dev libxml2-dev libxslt-dev

Repository https://ropensci.r-universe.dev

Date/Publication 2024-09-11 21:37:54 UTC

RemoteUrl https://github.com/ropensci/epubr

RemoteRef HEAD

RemoteSha 0d4097e9a3e87dd790c59230801356a0b8faf9da

Contents

count_words 2
epub 3
epub_cat 6
epub_head 7
epub_recombine 8
epub_reorder 9
epub_sift 10
epubr 11

Index **13**

`count_words` *Word count*

Description

Count the number of words in a string.

Usage

```
count_words(x, word_pattern = "[A-Za-z0-9&]", break_pattern = " |\n")
```

Arguments

`x`	character, a string containing words to be counted. May be a vector.
`word_pattern`	character, regular expression to match words. Elements not matched are not counted.
`break_pattern`	character, regular expression to split a string between words.

Details

This function estimates the number of words in strings. Words are first separated using `break_pattern`. Then the resulting character vector elements are counted, including only those that are matched by `word_pattern`. The approach taken is meant to be simple and flexible.

epub uses this function internally to estimate the number of words for each e-book section alongside the use of `nchar` for counting individual characters. It can be used directly on character strings and is convenient for applying with different regular expression pattern arguments as needed.

These two arguments are provided for control, but the defaults are likely good enough. By default, strings are split only on spaces and new line characters. The "words" that are counted in the resulting vector are those that contain any alphanumeric characters or the ampersand. This means for example that hyphenated words, acronyms and numbers displayed with digits, are all counted as words. The presence of any other characters does not negate that a word has been found.

Value

an integer

Examples

```
x <- " This   sentence will be counted to have:\n\n10 (ten) words."
count_words(x)
```

---

epub *Extract and read EPUB e-books*

Description

Read EPUB format e-books into a data frame using epub or extract EPUB archive files for direct use with `epub_unzip`.

Usage

```
epub(
  file,
  fields = NULL,
  drop_sections = NULL,
  chapter_pattern = NULL,
  encoding = "UTF-8",
  ...
)

epub_meta(file)

epub_unzip(file, exdir = tempdir())
```

Arguments

file	character, input EPUB filename. May be a vector for epub and epub_meta. Always a single file for epub_unzip.
fields	character, vector of metadata fields (data frame columns) to parse from metadata, if they exist. See details.
drop_sections	character, a regular expression pattern string to identify text sections (rows of nested text data frame) to drop.
chapter_pattern	character, a regular expression pattern string to attempt distinguishing nested data frame rows of chapter text entries from other types of entries.
encoding	character, defaults to "UTF-8".
...	additional arguments. With the exception of passing title (see details), currently developmental/unsupported.
exdir	for epub_unzip, extraction directory to place archive contents (files). It will be created if necessary.

Details

The primary function here is epub. It parses EPUB file metadata and textual content into a data frame. The output data frame has one row for each file in file. It has metadata in all columns except the data column, which is a column of nested data frames containing e-book text by book section (e.g., chapters). Both the primary and nested data frames are tibbles and safe to print to the console "as is".

Be careful if file is a long vector of many EPUB files. This could take a long time to process as well as could potentially use up all of your system RAM if you have far too many large books in one call to epub.

On a case by case basis, you can always select columns and filter rows of a resulting data frame for a single e-book subsequent to visual inspection. However, the optional arguments fields, drop_sections and chapter_pattern allow you to do some of this as part of the EPUB file reading process. You can ignore these arguments and do all your own post-processing of the resulting data frame, but if using these arguments, they are most likely to be useful for bulk e-book processing where file is a vector of like-formatted files.

Main columns: The fields argument can be used to limit the columns returned in the primary data frame. E.g., fields = c("title", "creator", "date", "identifier", "publisher", "file"). Some fields will be returned even if not in fields, such as data and title.

Ideally, you should already know what metadata fields are in the EPUB file. This is not possible for large collections with possibly different formatting. Note that when "file" is included in fields, the output will include a column of the original file names, in case this is different from the content of a source field that may be present in the metadata. So this field is always available even if not part of the file metadata.

Additionally, if there is no title field in the metadata, the output data frame will include a title column filled in with the same file names, unless you pass the additional optional title argument, e.g. title = "TitleFieldID" so that another field can me mapped to title. If supplying a

`title` argument that also does not match an existing field in the e-book, the output `title` column will again default to file names. File names are the fallback option because unlike e-book metadata fields, file names always exist and should also always be unique when performing vectorized reads over multiple books, ensuring that `title` can be a column in the output data frame that uniquely identifies different e-books even if the books did not have a `title` field in their metadata.

Columns of the nested data frames in `data` are fixed. Select from these in subsequent data frame manipulations.

Nested rows: The `chapter_pattern` argument may be helpful for bulk processing of similarly formatted EPUB files. This should be ignored for poorly formatted EPUB files or where there is inconsistent naming across an e-book collection. Like with `fields`, you should explore file metadata in advance or this argument will not be useful. If provided, a column `nchap` is added to the output data frame giving the guessed number of chapters. In the `data` column, the `section` column of the nested data frames will also be updated to reflect guessed chapters with new, consistent chapter IDs, always beginning with `ch` and ending with digits.

The `drop_sections` argument also uses regular expression pattern matching like `chapter_pattern` and operates on the same `section` column. It simply filters out any matched rows. This is useful for dropping rows that may pertain to book cover, copyright and acknowledgements pages, and other similar, clearly non-chapter text, e-book sections. An example that might work for many books could be `drop_sections = "^co(v|p)|^ack"`

Rows of the primary data frame are fixed. Filter or otherwise manipulate these in subsequent data frame manipulations. There is one row per file so filtering does not make sense to do as part of the initial file reading.

EPUB metadata: Use `epub_meta` to return a data frame of only the metadata for each file in `file`. This skips the reading of each file's text contents, strictly parsing the metadata. It returns a data frame with one row for each file and `n` columns where `n` is equal to the union of all fields identified across all files in `file`. Fields available for at least one e-book in `file` will return `NA` in that column for any row pertaining to an e-book that does not have that field in its metadata. If the metadata contains multiple entries for a field, such as multiple subjects or publication dates, they are collapsed using the pipe character.

Unzipping EPUB files: If using `epub_unzip` directly on individual EPUB files, this gives you control over where to extract archive files to and what to do with them subsequently. `epub` and `epub_meta` use `epub_unzip` internally to extract EPUB archive files to the R session temp directory (with `tempdir()`). You do not need to use `epub_unzip` directly prior to using these other functions. It is only needed if you want the internal files for some other purpose in or out of R.

Value

`epub` returns a data frame. `epub_unzip` returns nothing but extracts files from an EPUB file archive.

Examples

```
# Use a local example EPUB file included in the package
file <- system.file("dracula.epub", package = "epubr")
bookdir <- file.path(tempdir(), "dracula")
```

```
epub_unzip(file, exdir = bookdir) # unzip to directly inspect archive files
list.files(bookdir, recursive = TRUE)


epub_meta(file) # parse EPUB file metadata only

x <- epub(file) # parse entire e-book
x
x$data[[1]]

epub(file, fields = c("title", "creator"), drop_sections = "^cov")
```

`epub_cat` *Pretty printing of EPUB text*

Description

Print EPUB text to the console in a more readable format.

Usage

```
epub_cat(
  x,
  max_paragraphs = 10,
  skip = 0,
  paragraph_spacing = 1,
  paragraph_indent = 2,
  section_sep = "====",
  book_sep = "====\n====="
)
```

Arguments

`x`	a data frame returned by epub or a character string giving the EPUB filename(s).
`max_paragraphs`	integer, maximum number of paragraphs (non-empty lines) to `cat` to console.
`skip`	integer, number of paragraphs to skip.
`paragraph_spacing`	integer, number of empty lines between paragraphs.
`paragraph_indent`	integer, number of spaces to indent paragraphs.
`section_sep`	character, a string to indicate section breaks.
`book_sep`	character, separator shown between books when `x` has multiple rows (books).

Details

This function prints text from EPUB files to the console using `cat`. This is useful for quickly obtaining an overview of the book text parsed by epub that is easier to read that looking at strings in the table. `max_paragraphs` is set low by default to prevent accidentally printing entire books to the console. To print everything in `x`, set `max_paragraphs = NULL`.

Value

nothing is returned but a more readable format of the text content for books in `x` is printed to the console.

See Also

epub_head

Examples

```
file <- system.file("dracula.epub", package = "epubr")
d <- epub(file)
epub_cat(d, max_paragraphs = 2, skip = 147)
```

---

epub_head *Preview the first n characters*

Description

Preview the first n characters of each EPUB e-book section.

Usage

```
epub_head(x, n = 50)
```

Arguments

`x`	a data frame returned by epub or a character string giving the EPUB filename(s).
`n`	integer, first n characters to retain from each e-book section.

Details

This function returns a simplified data frame of only the unnested `section` and `text` columns of a data frame returned by epub, with the text included only up to the first `n` characters. This is useful for previewing the opening text of each e-book section to inspect for possible useful regular expression patterns to use for text-based section identification. For example, an e-book may not have meaningful section IDs that distinguish one type of book section from another, such as chapters from non-chapter sections, but the text itself may contain this information at or near the start of a section.

Value

a data frame.

See Also

epub_cat

Examples

```
file <- system.file("dracula.epub", package = "epubr")
epub_head(file)
```

---

epub_recombine *Recombine text sections*

Description

Split and recombine EPUB text sections based on regular expression pattern matching.

Usage

```
epub_recombine(data, pattern, sift = NULL)
```

Arguments

data	a data frame created by epub.
pattern	character, a regular expression.
sift	NULL or a named list of parameters passed to epub_sift. See details.

Details

This function takes a regular expression and uses it to determine new break points for the full e-book text. This is particularly useful when sections pulled from EPUB metadata have arbitrary breaks and the text contains meaningful breaks at random locations in various sections. epub_recombine collapses the text and then creates a new nested data frame containing new chapter/section labels, word counts and character counts, associated with the text based on the new break points.

Usefulness depends on the quality of the e-book. While this function exists to improve the data structure of e-book content parsed from e-books with poor metadata formatting, it still requires original formatting that will at least allow such an operation to be successful, specifically a consistent, non-ambiguous regular expression pattern. See examples below using the built in e-book dataset.

When used in conjunction with epub_sift via the sift argument, recombining and resifting is done recursively. This is because it is possible that sifting can create a need to rerun the recombine step in order to regenerate proper chapter indexing for the section column. However, recombining a second time does not lead to a need to resift, so recursion ends after one round regardless.

This is a convenient way to avoid the syntax:

epub_recombine([args]) %>% epub_sift([args]) %>% epub_recombine([args]).

Value

a data frame

See Also

epub_sift

Examples

```
file <- system.file("dracula.epub", package = "epubr")
x <- epub(file) # parse entire e-book
x$data[[1]] # note arbitrary section breaks (not between chapters)

pat <- "CHAPTER [IVX]+" # but a reliable pattern exists for new breaks
epub_recombine(x, pat) # not as expected; pattern also in table of contents

epub_recombine(x, pat, sift = list(n = 1000)) # sift low word-count sections
```

---

epub_reorder *Reorder sections*

Description

Reorder text sections in an e-book based on a user-provided function.

Usage

```
epub_reorder(data, .f, pattern)
```

Arguments

data	a data frame created by epub.
.f	a scalar function to determine a single row index based on a matched regular expression. It must take two strings, the text and the pattern, and return a single number. See examples.
pattern	regular expression passed to .f.

Details

Many e-books have chronologically ordered sections based on quality metadata. This results in properly book sections in the nested data frame. However, some poorly formatted e-books have their internal sections occur in an arbitrary order. This can be frustrating to work with when doing text analysis on each section and where order matters.

This function addresses this case by reordering the text sections in the nested data frame based on a user-provided function that re-indexes the data frame rows based on their content. In general, the approach is to find something in the content of each section that describes the section order. For

example, epub_recombine can use a regular expression to identify chapters. Taking this a step further, epub_reorder can use a function that works with the same information to reorder the rows.

It is enough in the former case to identify where in the text the pattern occurs. There is no need to extract numeric ordering from it. The latter takes more effort. In the example EPUB file included in epubr, chapters can be identified using a pattern of the word CHAPTER in capital letters followed by a space and then some Roman numerals. The user must provide a function that would parse the Roman numerals in this pattern so that the rows of the data frame can be reordered properly.

Value

a data frame

Examples

```
file <- system.file("dracula.epub", package = "epubr")
x <- epub(file) # parse entire e-book
x <- epub_recombine(x, "CHAPTER [IVX]+", sift = list(n = 1000)) # clean up

library(dplyr)
set.seed(1)
x$data[[1]] <- sample_frac(x$data[[1]]) # randomize rows for example
x$data[[1]]

f <- function(x, pattern) as.numeric(as.roman(gsub(pattern, "\\1", x)))
x <- epub_reorder(x, f, "^CHAPTER ([IVX]+).*")
x$data[[1]]
```

---

epub_sift *Sift EPUB sections*

Description

Sift out EPUB sections that have suspiciously low word or character count.

Usage

```
epub_sift(data, n, type = c("word", "char"))
```

Arguments

data	a data frame created by epub.
n	integer, minimum number of words or characters to retain a section.
type	character, "word" or "character".

Details

This function is like a sieve that lets small section rows fall through. Choose the minimum number of words or characters to accept as a meaningful section in the e-book worth retaining in the nested data frame, e.g., book chapters. Data frame rows pertaining to smaller sections are dropped.

This function is helpful for isolating meaningful content by removing extraneous e-book sections that may be difficult to remove by other methods when working with poorly formatted e-books. The EPUB file included in `epubr` is a good example of this. It does not contain meaningful section identifiers in its metadata. This creates a need to restructure the text table after reading it with `epub` by subsequently calling `epub_recombine`. However, some unavoidable ambiguity in this leads to many small sections appearing from the table of contents. These can then be dropped with `epub_sift`. See a more comprehensive in the `epub_recombine` documentation. A simpler example is shown below.

Value

a data frame

See Also

`epub_recombine`

Examples

```
file <- system.file("dracula.epub", package = "epubr")
x <- epub(file) # parse entire e-book
x$data[[1]]

x <- epub_sift(x, n = 3000) # drops last two sections
x$data[[1]]
```

`epubr` *epubr: Read EPUB File Metadata and Text*

Description

Provides functions supporting the reading and parsing of internal e-book content from EPUB files. The 'epubr' package provides functions supporting the reading and parsing of internal e-book content from EPUB files. E-book metadata and text content are parsed separately and joined together in a tidy, nested tibble data frame. E-book formatting is not completely standardized across all literature. It can be challenging to curate parsed e-book content across an arbitrary collection of e-books perfectly and in completely general form, to yield a singular, consistently formatted output. Many EPUB files do not even contain all the same pieces of information in their respective metadata. EPUB file parsing functionality in this package is intended for relatively general application to arbitrary EPUB e-books. However, poorly formatted e-books or e-books with highly uncommon formatting may not work with this package. There may even be cases where an EPUB file has DRM or some other property that makes it impossible to read with 'epubr'. Text is read 'as is' for the

most part. The only nominal changes are minor substitutions, for example curly quotes changed to straight quotes. Substantive changes are expected to be performed subsequently by the user as part of their text analysis. Additional text cleaning can be performed at the user's discretion, such as with functions from packages like 'tm' or 'qdap'.

Author(s)

Maintainer: Matthew Leonawicz <rpkgs@pm.me> (ORCID)

See Also

Useful links:

- https://docs.ropensci.org/epubr/
- https://github.com/ropensci/epubr
- Report bugs at https://github.com/ropensci/epubr/issues

Index

count_words, 2

epub, 3, *6, 7*
epub_cat, 6, *8*
epub_head, 7, *7*
epub_meta *(*epub*)*, 3
epub_recombine, 8, *11*
epub_reorder, 9
epub_sift, *8, 9*, 10
epub_unzip *(*epub*)*, 3
epubr, 11
epubr-package *(*epubr*)*, 11