Package 'cwbtools'

April 29, 2024

Type Package

Title Tools to Create, Modify and Manage 'CWB' Corpora

Version 0.4.2

Date 2024-04-28

Description The 'Corpus Workbench' ('CWB', <https://cwb.sourceforge.io/>) offers a classic and mature approach for working with large, linguistically and structurally annotated corpora. The 'CWB' is memory efficient and its design makes running queries fast, see Evert (2011) <https://eprints.lancs.ac.uk/id/eprint/62721>. The 'cwbtools' package offers pure 'R' tools to create indexed corpus files as well as high-level wrappers for the original 'C' implementation of 'CWB' as exposed by the 'RcppCWB' package (<https://CRAN.R-project.org/package=RcppCWB>). Additional functionality to add and modify annotations of corpora from within 'R' makes working with 'CWB' indexed corpora much more flexible and convenient. The 'cwbtools' package in combination with the 'R' packages 'RcppCWB' (<https://CRAN.R-project.org/package=RcppCWB>) and 'polmineR' (<https://CRAN.R-project.org/package=polmineR>) offers a lightweight infrastructure to support the combination of quantitative and qualitative approaches for working with textual data.

Imports data.table, R6, xml2, stringi, curl, RcppCWB (>= 0.6.3), pbapply, methods, tools, cli, jsonlite, httr, rstudioapi, zen4R (>= 0.9), lifecycle, fs

Suggests tm (>= 0.7.3), knitr, markdown, tokenizers (>= 0.2.1), tidytext, SnowballC, janeaustenr, testthat, rmarkdown, aws.s3, quanteda, dplyr

VignetteBuilder knitr

License GPL-3

Language en-US

Encoding UTF-8

URL https://github.com/PolMine/cwbtools

BugReports https://github.com/PolMine/cwbtools/issues

Collate 'CorpusData.R' 'corpus.R' 'cwb.R' 'cwbtools.R' 'directories.R' 'encoding.R' 'encode.R' 'ner.R' 'p_attribute.R' 'pkg.R' 'registry_file.R' 's_attribute.R' 'vrt.R' 'zenodo.R'

RoxygenNote 7.2.3

RdMacros lifecycle

NeedsCompilation no

Author Andreas Blaette [aut, cre], Christoph Leonhardt [aut]

Maintainer Andreas Blaette <andreas.blaette@uni-due.de>

Repository CRAN

Date/Publication 2024-04-28 22:20:06 UTC

R topics documented:

`cwbtools-package` *cwbtools-package*

Description

Tools to Create, Modify and Manage CWB Corpora.

Details

The *Corpus Workbench* (CWB) offers a classic approach for working with large, linguistically and structurally annotated corpora. Its design ensures memory efficiency and makes running queries fast (Evert and Hardie 2011). Technically, indexing and compressing corpora as suggested by Witten et al. (1999) is the approach implemented by the CWB (Christ 1994).

The C implementation of the CWB is mature and efficient. However, the convenience and flexibility of traditional CWB command line tools is limited. These tools are not portable across platforms, inhibiting the ideal of reproducible research.

The 'cwbtools' package combines portable pure R tools to create indexed corpus files and convenience wrappers for the original C implementation of CWB as exposed by the RcppCWB package. Additional functionality to add and modify annotations of corpora from within R makes working with CWB indexed corpora much more flexible. "Pure R" workflows to enrich corpora with annotations using standard NLP tools or generated manually can be implemented seamlessly and conveniently.

The *cwbtools* package is a companion of the RcppCWB and the polmineR package and is a building block of an infrastructure to support the combination of quantitative and qualitative approaches when working with textual data.

Author(s)

Andreas Blaette

References

Christ, Oliver (1994): "A Modular and Flexible Architecture for an Integrated Corpus Query System". *Proceedings of COMPLEX'94*, pp.23-32. (available online here)

Evert, Stefan and Andrew Hardie (2011): "Twenty-first century Corpus Workbench: Updating a query architecture for the new millennium." In: *Proceedings of the Corpus Linguistics 2011 conference*, University of Birmingham, UK. (available online here)

Witten, Ian H., Alistair Moffat and Timothy C. Bell (1999): *Managing Gigabytes: Compressing and Indexing Documents and Images.* 2nd edition. San Francisco et al.: Morgan Kaufmann.

`as.vrt` *Consolidate vrt files for CWB import.*

Description

Files resulting from tagging/annotation may violate the requirements of the Corpus Workbench (CWB). Consolidate the known issues the vrt files may cause.

Usage

```
as.vrt(x, replacements = list())
```

Arguments

`x`	a character vector providing a directory with vrt files
`replacements`	a list of character vectors (length 2 each) with regular expressions / replacements

Details

Known issues resulting from annotating files (with the treetagger in particular) are whitespace characters invalid for XML, XML elements at the end of a line rather than in a seperate line, characters invalid for XML (such as ampersands), inter alia.

Before doing respective corrections, the method tests whether there is any text at all in the files. Empty files (files that contain nothing but XML tags) are dropped.

`conll_get_regions` *Extract regions from NER annotations (CoNNL format).*

Description

Extract regions from NER annotations (CoNNL format).

Usage

```
conll_get_regions(x)
```

Arguments

`x`	A `data.frame`, a `data.table`, or any other object that can be coerced to a `data.table`. The input table is expected to have the columns "token" and "ner", and "cpos".

Examples

```
x <- data.frame(
  token = c(
    "Die",
    "Bundeskanzlerin",
    "Angela",
    "Merkel",
    "spricht",
    "im",
    "Bundestag",
    "zur",
    "Lage",
    "der",
    "Nation",
    "."
  ),
  ne = c("O", "O", "B-PERS", "I-PERS", "O", "O", "B-ORG", "O", "O", "O", "O", "O"),
  stringsAsFactors = FALSE
)
x[["cpos"]] <- 100L:(100L + nrow(x) - 1L)
tab <- conll_get_regions(x)
```

CorpusData *Manage Corpus Data and Encode CWB Corpus.*

Description

Manage Corpus Data and Encode CWB Corpus.

Manage Corpus Data and Encode CWB Corpus.

Details

See the CWB Encoding Tutorial on characters allowed for encoding attributes: "By convention, all attribute names must be lowercase (more precisely, they may only contain the characters a-z, 0-9, -, and _, and may not start with a digit). Therefore, the names of XML elements to be included in the CWB corpus must not contain any non-ASCII or uppercase letters." (section 2)

Import XML files.

Public fields

`chunktable` A `data.table` with column "id" (unique values), columns with metadata, and a column with text chunks.

`tokenstream` A `data.table` with a column "cpos" (corpus position), and columns with positional attributes, such as "word", "lemma", "pos", "stem".

`metadata` A `data.table` with a column "id", to link data with chunks/tokenstream, columns with document-level metadata, and a column "cpos_left" and "cpos_right", which can be generated using method `$add_corpus_positions()`.

`sentences` A `data.table`.

`named_entities` A `data.table`.

Methods

Public methods:

- `CorpusData$new()`
- `CorpusData$print()`
- `CorpusData$tokenize()`
- `CorpusData$import_xml()`
- `CorpusData$add_corpus_positions()`
- `CorpusData$purge()`
- `CorpusData$encode()`
- `CorpusData$clone()`

Method `new()`**:** Initialize a new instance of class `CorpusData`.

Usage:

```
CorpusData$new()
```

Returns: A class `CorpusData` object.

Method `print()`**:** Print summary of `CorpusData` object.

Usage:

```
CorpusData$print()
```

Method `tokenize()`**:** Simple tokenization of text in chunktable.

Usage:

```
CorpusData$tokenize(..., verbose = TRUE, progress = TRUE)
```

Arguments:

`...` Arguments that are passed into `tokenizers::tokenize_words()`.

`verbose` A logical value, whether to be verbose.

`progress` A logical value, whether to show progress bar.

Method `import_xml()`**:**

Usage:

```
CorpusData$import_xml(
  filenames,
  body = "//body",
  meta = NULL,
  mc = NULL,
  progress = TRUE
)
```

Arguments:

`filenames` A vector of files to process.

`body` An xpath expression defining the body of the XML document.

`meta` A named character vector with XPath expressions.

`mc` A numeric/integer value, number of cores to use.

`progress` A logical value, whether to show progress bar.

Returns: The `CorpusData` object is returned invisibly.

Method `add_corpus_positions()`**:** Add column 'cpos' to tokenstream and columns 'cpos_left' and 'cpos_right' to metadata.

Usage:

```
CorpusData$add_corpus_positions(verbose = TRUE)
```

Arguments:

`verbose` A logical value, whether to be verbose.

Method `purge()`**:** Remove patterns from chunkdata that are known to cause problems. This is done most efficiently at the chunkdata level of data preparation as the length of the character vector to handle is much smaller than when tokenization/annotation has been performed.

Usage:

```
CorpusData$purge(
  replacements = list(c("^\\s*<.*?>\\s*$", ""), c("'", "'"))
)
```

Arguments:

`replacements` A list of length-two character vectors with regular expressions and replacements.

Method `encode()`**:** Encode corpus. If the corpus already exists, it will be removed.

Usage:

```
CorpusData$encode(
  corpus,
  p_attributes = "word",
  s_attributes = NULL,
  encoding,
  registry_dir = Sys.getenv("CORPUS_REGISTRY"),
  data_dir = NULL,
  method = c("R", "CWB"),
  verbose = TRUE,
  compress = FALSE,
  reload = TRUE,
  quietly = TRUE
)
```

Arguments:

`corpus` The name of the CWB corpus.

`p_attributes` Positional attributes.

`s_attributes` Columns that will be encoded as structural attributes.

`encoding` Encoding/charset of the CWB corpus.

`registry_dir` Corpus registry, the directory where registry files are stored.

`data_dir` Directory where to create directory for indexed corpus files.

`method` Either "R" or "CWB".

`verbose` A logical value, whether to be verbose.

`compress` A logical value, whether to compress corpus.

`reload` A `logical` value, whether to reload corpus.

`quietly` A `logical` value passed into `RcppCWB::cwb_makeall()`, `RcppCWB::cwb_huffcode()` and `RcppCWB::cwb_compress_rdx` to control verbosity of these functions.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CorpusData$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(RcppCWB)
library(data.table)

# this example relies on the R method to write data to disk, there is also a method "CWB"
# that relies on CWB tools to generate the indexed corpus. The CWB can downloaded
```

```
# and installed within the package by calling cwb_install()

# create temporary registry file so that data in RcppCWB package can be used

registry_rcppcwb <- system.file(package = "RcppCWB", "extdata", "cwb", "registry")
registry_tmp <- fs::path(tempdir(), "registry")
if (!dir.exists(registry_tmp)) dir.create(registry_tmp)
r <- registry_file_parse("REUTERS", registry_dir = registry_rcppcwb)
r[["home"]] <- system.file(package = "RcppCWB", "extdata", "cwb", "indexed_corpora", "reuters")
registry_file_write(r, corpus = "REUTERS", registry_dir = registry_tmp)

# decode structural attribute 'places'

s_attrs_places <- RcppCWB::s_attribute_decode(
  corpus = "REUTERS",
 data_dir = system.file(package = "RcppCWB", "extdata", "cwb", "indexed_corpora", "reuters"),
  s_attribute = "places", method = "R"
)
s_attrs_places[["id"]] <- 1L:nrow(s_attrs_places)
setnames(s_attrs_places, old = "value", new = "places")

# decode positional attribute 'word'

tokens <- apply(s_attrs_places, 1, function(row){
  ids <- cl_cpos2id(
    corpus = "REUTERS", cpos = row[1]:row[2],
    p_attribute = "word", registry = registry_tmp
  )
  cl_id2str(corpus = "REUTERS", id = ids, p_attribute = "word", registry = registry_tmp)
})
tokenstream <- rbindlist(
lapply(
  1L:length(tokens),
  function(i) data.table(id = i, word = tokens[[i]]))
  )
tokenstream[["cpos"]] <- 0L:(nrow(tokenstream) - 1L)

# create CorpusData object (see vignette for further explanation)

CD <- CorpusData$new()
CD$tokenstream <- as.data.table(tokenstream)
CD$metadata <- as.data.table(s_attrs_places)

# Remove temporary registry with home dir still pointing to RcppCWB data dir
# to prevent data from being deleted
file.remove(fs::path(registry_tmp, "reuters"))
file.remove(registry_tmp)

# create temporary directories (registry directory and one for indexed corpora)

registry_tmp <- fs::path(tempdir(), "registry")
data_dir_tmp <- fs::path(tempdir(), "data_dir")
if (!dir.exists(registry_tmp)) dir.create(registry_tmp)
```

```
if (!dir.exists(data_dir_tmp)) dir.create(data_dir_tmp)

CD$encode(
  corpus = "REUTERS", encoding = "utf8",
  p_attributes = "word", s_attributes = "places",
  registry_dir = registry_tmp, data_dir = data_dir_tmp,
  method = "R"
)
reg <- registry_data(name = "REUTERS", id = "REUTERS", home = data_dir_tmp, p_attributes = "word")
registry_file_write(data = reg, corpus = "REUTERS", registry_dir = registry_tmp)

# see whether it works

cl_cpos2id(corpus = "REUTERS", p_attribute = "word", cpos = 0L:4049L, registry = registry_tmp)
```

---

corpus_install *Install and manage corpora.*

Description

Utility functions to assist the installation and management of indexed CWB corpora.

Usage

```
corpus_install(
  pkg = NULL,
  repo = "https://PolMine.github.io/drat/",
  tarball = NULL,
  doi = NULL,
  checksum = NULL,
  lib = .libPaths()[1],
  registry_dir,
  corpus_dir,
  ask = interactive(),
  load = TRUE,
  verbose = TRUE,
  user = NULL,
  password = NULL,
  ...
)

corpus_packages()

corpus_rename(
  old,
  new,
  registry_dir = Sys.getenv("CORPUS_REGISTRY"),
  verbose = TRUE
```

```
)

corpus_remove(corpus, registry_dir, ask = interactive(), verbose = TRUE)

corpus_as_tarball(
  corpus,
  registry_dir,
  data_dir = registry_file_parse(corpus, registry_dir)[["home"]],
  tarfile,
  verbose = TRUE
)

corpus_copy(
  corpus,
  registry_dir,
  data_dir = registry_file_parse(corpus, registry_dir)[["home"]],
  registry_dir_new = fs::path(tempdir(), "cwb", "registry"),
  data_dir_new = fs::path(tempdir(), "cwb", "indexed_corpora", tolower(corpus)),
  remove = FALSE,
  verbose = interactive(),
  progress = TRUE
)

corpus_recode(
  corpus,
  registry_dir = Sys.getenv("CORPUS_REGISTRY"),
  data_dir = registry_file_parse(corpus, registry_dir)[["home"]],
  skip = character(),
  to = c("latin1", "UTF-8"),
  verbose = TRUE
)

corpus_testload(
  corpus,
  registry_dir = Sys.getenv("CORPUS_REGISTRY"),
  verbose = TRUE
)

corpus_get_version(corpus, registry_dir = Sys.getenv("CORPUS_REGISTRY"))

corpus_reload(corpus, registry_dir, verbose = TRUE)
```

Arguments

pkg	Name of a package (length-one character vector).
repo	URL of the repository.
tarball	URL, S3-URI or local filename of a tarball with a CWB indexed corpus. If NULL (default) and argument doi is stated, the whereabouts of a corpus tarball will be

	derived from DOI.
doi	The DOI (Digital Object Identifier) of a corpus deposited at Zenodo (e.g. "10.5281/zenodo.3748858".)
checksum	A length-one character vector with a MD5 checksum to check for the integrity of a downloaded tarball. If the tarball is downloaded from Zenodo by stating a DOI (argument doi), the checksum included in the metadata for the record is used for the check.
lib	Directory for R packages, defaults to .libPaths()[1].
registry_dir	The corpus registry directory. If missing, the result of cwb_registry_dir().
corpus_dir	The directory that contains the data directories of indexed corpora. If missing, the value of cwb_corpus_dir() will be used.
ask	A logical value, whether to ask user for confirmation before removing a corpus.
load	A logical value, whether to load corpus after installation.
verbose	Logical, whether to be verbose.
user	A user name that can be specified to download a corpus from a password protected site.
password	A password that can be specified to download a corpus from a password protected site.
...	Further parameters that will be passed into download.file(), if tarball is specified.
old	Name of the (old) corpus.
new	Name of the (new) corpus.
corpus	The ID of a CWB indexed corpus (in upper case).
data_dir	The data directory where the files of the CWB corpus live.
tarfile	Filename of tarball.
registry_dir_new	Target directory with for (new) registry files.
data_dir_new	Target directory for corpus files.
remove	A logical value, whether to remove orginal files after having created the copy.
progress	Logical, whether to show a progress bar.
skip	A character vector with s_attributes to skip.
to	Character string describing the target encoding of the corpus.

Details

A CWB corpus consists a set of binary files with corpus data kept together in a data directory, and a registry file, which is a plain test file that details the corpus id, corpus properties, structural and positional attributes. The registry file also specifies the path to the corpus data directory. Typically, the registry directory and a corpus directory with the data directories for individual corpora are within one parent folder (which might be called "cwb" by default). See the following stylized directory structure.

```
.
|- registry/
|  |- corpus1
|  +- corpus2
|
+ indexed_corpora/
  |- corpus1/
  |  |- file1
  |  |- file2
  |  +- file3
  |
  +- corpus2/
     |- file1
     |- file2
     +- file3
```

The `corpus_install()` function will assist the installation of a corpus. The following scenarios are offered:

- If argument `tarball` is a local tarball, the tarball will be extracted and files will be moved.
- If `tarball` is a URL, the tarball will be downloaded from the online location. It is possible to state user credentials using the arguments `user` and `password`. Then the aforementioned installation (scenario 1) is executed. If argument `pkg` is the name of an installed package, corpus files will be moved into this package.
- If argument `doi` is Document Object Identifier (DOI), the URL from which a corpus tarball can be downloaded is derived from the information available at that location. The tarball is downloaded and the corpus installed. If argument `pkg` is defined, files will be moved into a R package, the syste registry and corpus directories are used otherwise. Note that at this stage, it is assumed that the DOI has been awarded by Zenodo
- If argument `pkg` is provided and `tarball` is `NULL`, corpora included in the package will be installed as system corpora, using the storage location specified by `registry_dir`.

If the corpus to be installed is already available, a dialogue will ask the user whether an existing corpus shall be deleted and installed anew, if argument `ask` is `TRUE`.

`corpus_packages()` will detect the packages that include CWB corpora. Note that the directory structure of all installed packages is evaluated which may be slow on network-mounted file systems.

`corpus_rename()` will rename a corpus, affecting the name of the registry file, the corpus id, and the name of the directory where data files reside.

`corpus_remove()` can be used to delete a corpus.

`corpus_as_tarball()` will create a tarball (.tar.gz-file) with two subdirectories. The 'registry' subdirectory will host the registry file for the tarred corpus. The data files will be put in a subdirectory with the corpus name in the 'indexed_corpora' subdirectory.

`corpus_copy()` will create a copy of a corpus (useful for experimental modifications, for instance).

`corpus_get_version` parses the registry file and derives the corpus version number from the corpus properties. The return value is a `numeric_version` class object. The corpus version is expected to follow semantic versioning (three digits, e.g. '0.8.1'). If the corpus version has another format or if it is not available, the return value is `NA`.

corpus_reload() will unload a corpus if necessary and reload it. Useful to make new features of a corpus available after modification. Returns logical value TRUE if succesful, FALSE if not.

Value

Logical value TRUE if installation has been successful, or FALSE if not.

See Also

For managing registry files, see registry_file_parse for switching to a packaged corpus.

Examples

```
registry_file_new <- fs::path(tempdir(), "cwb", "registry", "reuters")
if (file.exists(registry_file_new)) file.remove(registry_file_new)
corpus_copy(
  corpus = "REUTERS",
  registry_dir = system.file(package = "RcppCWB", "extdata", "cwb", "registry"),
  data_dir = system.file(
    package = "RcppCWB",
    "extdata", "cwb", "indexed_corpora", "reuters"
  )
)
unlink(fs::path(tempdir(), "cwb"), recursive = TRUE)
corpus <- "REUTERS"
pkg <- "RcppCWB"
s_attr <- "places"
Q <- '"oil"'

registry_dir_src <- system.file(package = pkg, "extdata", "cwb", "registry")
data_dir_src <- system.file(package = pkg, "extdata", "cwb", "indexed_corpora", tolower(corpus))

registry_dir_tmp <- fs::path(tempdir(), "cwb", "registry")
registry_file_tmp <- fs::path(registry_dir_tmp, tolower(corpus))
data_dir_tmp <- fs::path(tempdir(), "cwb", "indexed_corpora", tolower(corpus))

if (file.exists(registry_file_tmp)) file.remove(registry_file_tmp)
if (!dir.exists(data_dir_tmp)){
   dir.create(data_dir_tmp, recursive = TRUE)
} else {
  if (length(list.files(data_dir_tmp)) > 0L)
    file.remove(list.files(data_dir_tmp, full.names = TRUE))
}

corpus_copy(
  corpus = corpus,
  registry_dir = registry_dir_src,
  data_dir = data_dir_src,
  registry_dir_new = registry_dir_tmp,
  data_dir_new = data_dir_tmp
)
```

```
RcppCWB::cl_charset_name(corpus = corpus, registry = registry_dir_tmp)

corpus_recode(
  corpus = corpus,
  registry_dir = registry_dir_tmp,
  data_dir = data_dir_tmp,
  to = "UTF-8"
)

RcppCWB::cl_delete_corpus(corpus = corpus, registry = registry_dir_tmp)
RcppCWB::cqp_initialize(registry_dir_tmp)
RcppCWB::cl_charset_name(corpus = corpus, registry = registry_dir_tmp)

n_strucs <- RcppCWB::cl_attribute_size(
  corpus = corpus, attribute = s_attr, attribute_type = "s", registry = registry_dir_tmp
)
strucs <- 0L:(n_strucs - 1L)
struc_values <- RcppCWB::cl_struc2str(
  corpus = corpus, s_attribute = s_attr, struc = strucs, registry = registry_dir_tmp
)
speakers <- unique(struc_values)

Sys.setenv("CORPUS_REGISTRY" = registry_dir_tmp)
if (RcppCWB::cqp_is_initialized()) RcppCWB::cqp_reset_registry() else RcppCWB::cqp_initialize()
RcppCWB::cqp_query(corpus = corpus, query = Q)
cpos <- RcppCWB::cqp_dump_subcorpus(corpus = corpus)
ids <- RcppCWB::cl_cpos2id(
  corpus = corpus, p_attribute = "word", registry = registry_dir_tmp, cpos = cpos
)
str <- RcppCWB::cl_id2str(
  corpus = corpus, p_attribute = "word", registry = registry_dir_tmp, id = ids
)
unique(str)

unlink(fs::path(tempdir(), "cwb"), recursive = TRUE)
```

`cwb_corpus_dir` *Manage directories for indexed corpora*

Description

The Corpus Workbench (CWB) stores the binary files for structural and positional attributes in an individual 'data directory' (referred to by argument `data_dir`) for each corpus. The data directories will typically be subdirectories of a parent directory called 'corpus directory' (argument `corpus_dir`). Irrespective of the location of the data directories, all corpora available on a machine are described by so-called (plain text) registry files stored in a so-called 'registry directory' (referred to by argument `registry_dir`). The functionality to manage theses directories is used as auxiliary functionality by higher-level functionality to download and install corpora.

Usage

```
cwb_corpus_dir(registry_dir, verbose = TRUE)

cwb_registry_dir(verbose = TRUE)

cwb_directories(registry_dir = NULL, corpus_dir = NULL, verbose = TRUE)

create_cwb_directories(prefix = "~/cwb", ask = interactive(), verbose = TRUE)

use_corpus_registry_envvar(registry_dir)
```

Arguments

registry_dir	Path to the directory with registry files.
verbose	A logical value, whether to output status messages.
corpus_dir	Path to the directory with data directories for corpora.
prefix	The base path that will be prefixed
ask	A logical value, whether to prompt user before creating directories.

Details

cwb_corpus_dir will make a plausible suggestion for a corpus directory where data directories for corpora reside. The procedure requires that the registry directory (argument registry_dir) is known. If the argument registry_dir is missing, the registry directory will be guessed by calling cwb_registry_dir. The heuristic to detect the corpus directory is as follows: First, directories in the parent directory of the registry directory that contain "corpus" or "corpora" are suggested. If this does not yield a result, the data directories stated in the registry files are evaluated. If there is one unique parent directory of data directories (after removing temporary directories and directories within packages), this unique directory is suggested. cwb_corpus_dir will return a length-one character vector with the path of the suggested corpus directory, or NULL if the heuristic does not yield a result.

cwb_registry_dir() will return return the system registry directory. By default, the environment variable CORPUS_REGISTRY defines the system registry directory. If the polmineR-package is loaded, a temporary registry directory is used, replacing the system registry directory. In this case, cwb_registry_dir() will retrieve the directory from the option 'polmineR.corpus_registry'. The return value is a length-one character vector or NULL, if no registry directory can be detected.

cwb_directories will return a named character vector with the registry directory and the corpus directory.

create_cwb_directories will create a 'registry' and an 'indexed_corpora' directory as subdirectories of the directory indicated by argument prefix. Argument ask indicates whether to create directories, and whether user feedback is asked for before creating the directories. The function returns a named character vector with the registry and the corpus directory.

use_corpus_registry_envvar() is a convenience function that will assist users to define the environment variable CORPUS_REGSITRY in the .Renviron-file. making it available across sessions. The function is intended to be used in an interactive R session. An error is thrown if this is not the

case. The user will be prompted whether the cwbtools package shall take care of creating / modifying the .Renviron-file. If not, the file will be opened for manual modification with some instructions shown in the terminal.

`cwb_install` *Utilities to install the Corpus Workbench (CWB)*

Description

The CWB comprises a set of command line tools for corpus preparation and management. Functionality for installing and managing these Tools.

Usage

```
cwb_install(
  url_cwb = cwb_get_url(),
  md5 = attr(url_cwb, "md5"),
  cwb_dir = fs::path(fs::path_temp(), "cwb"),
  verbose = TRUE
)

cwb_get_url()

cwb_get_bindir(bindir = Sys.getenv("CWB_BINDIR"), verbose = TRUE)

cwb_is_installed()
```

Arguments

`url_cwb`	URL for downloading the CWB.
`md5`	The md5 checksum of the compressed file to be downloaded.
`cwb_dir`	The directory where the CWB shall be installed.
`verbose`	Logical, whether to show progress messages.
`bindir`	The directory with CWB binaries.

Details

Use `cwb_install()` to download and install CWB binaries (v3.5) from SourceForge. If successful, `cwb_install()` returns the directory of the CWB, otherwiese `NULL`. For the installation on macOS and Linux, see `https://cwb.sourceforge.io/install.php`.

`cwb_get_url()` will return the URL for downloading the appropriate binary (Linux / macOS) of the CWB (v3.5), or the source tarball (Linux). The md5 checksum of the file to be downloaded is part of the return value as "md5" attribute.

`cwb_get_bindir()` detects the directory with the cwb command line programs. Defaults to using the value of the environment variable "CWB_BINDIR". If unset, the value of `cwb-config --bindir` is used. Returns `NULL` if CWB installation is not found.

`cwb_is_installed()` will check whether the CWB is installed.

encode *Encode CWB Corpus.*

Description

[Experimental]

Usage

```
encode(x, ...)

## S4 method for signature 'data.frame'
encode(
  x,
  corpus,
  s_attributes = NULL,
  encoding = "utf8",
  registry_dir = fs::path(tempdir(), "cwb_registry"),
  data_dir = fs::path(tempdir(), "cwb_data_dir", tolower(corpus)),
  properties = c(),
  method = c("R", "CWB"),
  verbose = TRUE,
  compress = FALSE,
  reload = TRUE,
  quietly = TRUE
)
```

Arguments

x	A `data.frame` or an object inheriting from `data.frame` (such as `tibble`, `data.table`).
...	Further arguments (unused).
corpus	ID of the CWB corpus to create.
s_attributes	A `list` of `data.frame` objects with columns 'cpos_left' and 'cpos_right' and columns with s-attributes, the names of which will serve as names of s-attributes. It `s_attributes` is a `data.frame`, it will be coerced to a `list`.
encoding	Encoding as defined in the charset corpus property of the registry file for the corpus ('latin1' to 'latin9', and 'utf8').
registry_dir	Registry directory.
data_dir	The data directory for the binary files of the corpus.
properties	A named `character` vector with corpus properties that will be added to the registry file describing the corpus. Names of the vector indicate a property (such as "version") and the values of the vector the values of a corpus property.
method	Either 'CWB' or 'R', defaults to 'R'. See section 'Details'.
verbose	A `logical` value, whether to output progress messages.

compress	A logical value, whether to run RcppCWB::cwb_huffcode() and RcppCWB::cwb_compress_rdx() (method 'R'), or command line tools cwb-huffcode and cwb-compress-rdx (method 'CWB'). Defaults to FALSE as compression is not stable on Windows.
reload	A logical value, whether to reload the corpus to make it immediatedly available.
quietly	A logical value passed into RcppCWB::cwb_makeall(), RcppCWB::cwb_huffcode() and RcppCWB::cwb_compress_rdx to control verbosity of these functions.

Examples

```
# This is an example we run conditionally as packages are suggested.

dplyr_available <- requireNamespace("dplyr")
tidytext_available <- requireNamespace("tidytext")
quanteda_available <- requireNamespace("quanteda")

if (dplyr_available && tidytext_available && quanteda_available){

library(dplyr) # pipe would not be available otherwise
library(tidytext)

registry_tmp <- fs::path(tempdir(), "cwb_registry")
dir.create(registry_tmp)

tidydata <- quanteda::data_char_ukimmig2010 %>%
   as.data.frame() %>%
   as_tibble(rownames = "party") %>%
   rename(`text` = ".")

tokenstream <- tidydata %>%
   unnest_tokens(word, text, to_lower = FALSE, strip_punct = FALSE) %>%
   mutate(cpos = 0L:(nrow(.) - 1L))

metadata <- tokenstream %>%
  group_by(party) %>%
  summarise(cpos_left = min(cpos), cpos_right = max(cpos))

tokenstream %>%
  select(-cpos, -party) %>%
  encode(
    corpus = "UKIMMIG2010",
    s_attributes = metadata,
    properties = c(lang = "en")
  )

}
```

---

get_encoding *Get Encoding of Character Vector.*

Description

Get Encoding of Character Vector.

Usage

```
get_encoding(x, verbose = FALSE)
```

Arguments

x	a character vector
verbose	logical, whether to output messages

---

pkg_utils *Create and manage packages with corpus data.*

Description

Putting CWB indexed corpora into R data packages is a convenient way to ship and share corpora, and to keep documentation and supplementary functionality with the data.

[Deprecated]

Usage

```
pkg_create_cwb_dirs(pkg = ".", verbose = TRUE)

pkg_add_corpus(
  pkg = ".",
  corpus,
  registry = Sys.getenv("CORPUS_REGISTRY"),
  verbose = TRUE
)

pkg_add_configure_scripts(pkg = ".")

pkg_add_description(
  pkg = ".",
  package = NULL,
  version = "0.0.1",
  date = Sys.Date(),
  author,
  maintainer = NULL,
  description = "",
  license = "",
  verbose = TRUE
)
```

```
pkg_add_creativecommons_license(
  pkg = ".",
  license = "CC-BY-NC-SA",
 file = system.file(package = "cwbtools", "txt", "licenses", "CC_BY-NC-SA_3.0.txt")
)

pkg_add_gitattributes_file(pkg = ".")
```

Arguments

pkg	Path to directory of data package or package name.
verbose	A logical value, whether to be verbose.
corpus	Name of the CWB corpus to insert into the package.
registry	Registry directory.
package	The package name (character), may not include special chars, and no underscores ('_').
version	The version number of the corpus (defaults to "0.0.1")
date	The date of creation, defaults to Sys.Date().
author	The author of the package, either character vector or object of class person.
maintainer	Maintainer, R package style, either character vector or person.
description	description of the data package.
license	The license.
file	Path to file with fulltext of Creative Commons license.

Details

pkg_creage_cwb_dirs will create the standard directory structure for storing registry files and indexed corpora within a package (./inst/extdata/cwb/registry and ./inst/extdata/cwb/indexed_corpora, respectively).

pkg_add_corpus will add the corpus described in registry directory to the package defined by pkg.

add_configure_script will add standardized and tested configure scripts configure for Linux and macOS, and configure.win for Windows to the top level directory of the data package, and file setpaths.R to tools subdirectory. The configuration mechanism ensures that the data directory is specified correctly in the registry files during the installation of the data package.

pkg_add_description will add a description file to the package.

pkg_add_creativecommons_license will license information to the DESCRIPTION file, and move file LICENSE to top level directory of the package.

pkg_add_gitattributes_file will add a file '.gitattributes' to the package. The file defines types of files that will be tracked by Git LFS, i.e. they will not be under conventional version control. This is suitable for large binary files, which is the scenario applicable for indexed corpus data.

References

Blätte, Andreas (2018). "Using Data Packages to Ship Annotated Corpora of Parliamentary Protocols: The GermaParl R Package", *ParlaCLARIN 2018 Workshop Proceedings*, available online here.

Examples

```
pkgdir <- fs::path_temp()
pkg_create_cwb_dirs(pkg = pkgdir)
pkg_add_description(
  pkg = pkgdir,
  package = "reuters",
  author = "cwbtools",
  description = "Reuters data package"
 )
pkg_add_corpus(
  pkg = pkgdir, corpus = "REUTERS",
  registry = system.file(package = "RcppCWB", "extdata", "cwb", "registry")
)
pkg_add_gitattributes_file(pkg = pkgdir)
pkg_add_configure_scripts(pkg = pkgdir)
pkg_add_creativecommons_license(pkg = pkgdir)
```

---

p_attribute_encode *Encode Positional Attribute(s).*

Description

Generate positional attribute from a character vector of tokens (the token stream).

Usage

```
p_attribute_encode(
  token_stream,
  p_attribute = "word",
  registry_dir,
  corpus,
  data_dir,
  method = c("R", "CWB"),
  verbose = TRUE,
  quietly = FALSE,
  encoding = get_encoding(token_stream),
  compress = FALSE,
  reload = TRUE
)

p_attribute_recode(
  data_dir,
  p_attribute,
  from = c("UTF-8", "latin1"),
  to = c("UTF-8", "latin1")
)

p_attribute_rename(
```

```
  corpus,
  old,
  new,
  registry_dir,
  verbose = TRUE,
  dryrun = FALSE
)
```

Arguments

token_stream	A character vector with the tokens of the corpus. The maximum length is 2 147 483 647 (2^31 - 1); a warning is issued if this threshold is exceeded. See the CWB Encoding Tutorial for size limitations of corpora. May also be a file.
p_attribute	The positional attribute to create - a character vector containing only lowercase ASCII characters (a-z), digits (0-9), -, and _: No non-ASCII or uppercase letters allowed. If method is "R", only one positional attribute can be encoded at a time. If method is "CWB", more than one p-attribute allowed.
registry_dir	Registry directory.
corpus	ID of the CWB corpus to create.
data_dir	The data directory for the binary files of the corpus.
method	Either 'CWB' or 'R', defaults to 'R'. See section 'Details'.
verbose	A logical value, whether to output progress messages.
quietly	A logical value passed into RcppCWB::cwb_makeall(), RcppCWB::cwb_huffcode() and RcppCWB::cwb_compress_rdx to control verbosity of these functions.
encoding	Encoding as defined in the charset corpus property of the registry file for the corpus ('latin1' to 'latin9', and 'utf8').
compress	A logical value, whether to run RcppCWB::cwb_huffcode() and RcppCWB::cwb_compress_rdx() (method 'R'), or command line tools cwb-huffcode and cwb-compress-rdx (method 'CWB'). Defaults to FALSE as compression is not stable on Windows.
reload	A logical value that defaults to TRUE to ensure that all features are available.
from	Character string describing the current encoding of the attribute.
to	Character string describing the target encoding of the attribute.
old	A character vector with p-attributes to be renamed.
new	A character vector with new names of p-attributes. The vector needs to have the same length as vector old.
dryrun	A logical value, whether to suppress actual renaming operation for inspecting output messages

Details

Four steps generate the binary CWB corpus data format for positional attributes: (1) Encode the token stream of the corpus, (2) create index files, (3) compress token stream and (4) compress index files. Whereas steps 1 and 2 are required to make a corpus work, steps 3 and 4 are optional yet useful to reduce disk usage and improve performance. See the CQP Corpus Encoding Tutorial (sections 2-4) for an explanation of the procedure.

p_attribute_encode() offers an R and a CWB implementation controlled by argument method. When choosing method 'R', the token stream is encoded in 'pure R', then the C implementation of CWB functionality as exposed to R via the RcppCWB package is used (functions RcppCWB::cwb_makeall() for indexing, RcppCWB::cwb_huffcode() and RcppCWB::cwb_compress_rdx() for compression). When choosing method 'CWB', the token stream is written to disk, then CWB command line utilities 'cwb-encode', cwb-makeall', 'cwb-huffcode' and 'cwb-compress-rdx' are called using system2(). The CWB-method requires an installation of the 'CWB'. The cwb_install() function will download and # install the CWB command line tools within the package. The 'CWB'-method is still supported as it is used in the test suite of the packaage. The 'R'-method is robust and is recommended.

p_attribute_recode() will recode the values in the avs-file and change the attribute value index in the avx file. The rng-file remains unchanged. The registry file remains unchanged, and it is highly recommended to consider s_attribute_recode() as a helper for corpus_recode() that will recode all s-attributes, all p-attributes, and will reset the encoding in the registry file.

Function p_attribute_rename() can be used to rename a positional attribute. Note that the corpus is not refreshed (unloaded, re-loaded), so it may be necessary to restart R for changes to become effective.

Value

TRUE is returned invisibly, if encoding has been successful. FALSE indicates an error has occurred.

Author(s)

Christoph Leonhardt, Andreas Blaette

Examples

```
# In this example, we follow a "pure R" approach.
library(dplyr)

reu <- system.file(package = "RcppCWB", "extdata", "examples", "reuters.txt")
tokens <- readLines(reu)

# Create new (and empty) directory structure

registry_tmp <- fs::path(tempdir(), "registry")
data_dir_tmp <- fs::path(tempdir(), "data_dir", "reuters")

if (dir.exists(registry_tmp)) unlink(registry_tmp, recursive = TRUE)
if (dir.exists(data_dir_tmp)) unlink(data_dir_tmp, recursive = TRUE)

dir.create(registry_tmp)
dir.create(data_dir_tmp, recursive = TRUE)

# Encode token stream (without compression)

p_attribute_encode(
  corpus = "reuters",
  token_stream = tokens,
```

```
  p_attribute = "word",
  data_dir = data_dir_tmp,
  registry_dir = registry_tmp,
  method = "R",
  compress = FALSE,
  quietly = TRUE,
  encoding = "utf8"
)

# Augment registry file

registry_file_parse(corpus = "REUTERS", registry_dir = registry_tmp) %>%
  registry_set_name("Reuters Sample Corpus") %>%
  registry_set_property("charset", "utf8") %>%
  registry_set_property("language", "en") %>%
  registry_set_property("build_date", as.character(Sys.Date())) %>%
  registry_file_write()

# Run query as a test

library(RcppCWB)

cqp_query(corpus = "REUTERS", query = '[]{3} "oil" []{3};')
regions <- cqp_dump_subcorpus(corpus = "REUTERS")

kwic <- apply(
  regions, 1,
  function(region){
    ids <- cl_cpos2id(
      "REUTERS",
      p_attribute = "word",
      registry = registry_tmp,
      cpos = region[1]:region[2]
    )
    words <- cl_id2str(
      corpus = "REUTERS",
      p_attribute = "word",
      registry = registry_tmp,
      id = ids
    )
    paste0(words, collapse = " ")
  }
)
kwic[1:10]
```

`registry_file_parse` *Parse and create registry files.*

Description

A set of functions to parse, create and write registry files.

Usage

```
registry_file_parse(corpus, registry_dir = Sys.getenv("CORPUS_REGISTRY"))

registry_file_compose(x)

registry_data(
  name,
  id,
  home,
  info = fs::path(home, ".info"),
  properties = c(charset = "utf-8"),
  p_attributes,
  s_attributes = character()
)

registry_file_write(
  data,
  corpus,
  registry_dir = Sys.getenv("CORPUS_REGISTRY"),
  ...
)

registry_set_property(data, property, value)

registry_set_info(data, info_file)

registry_set_name(data, name)
```

Arguments

corpus	A CWB corpus indicated by a length-one character vector.
registry_dir	Directory with registry files.
x	An object of class registry_data.
name	Long descriptive name of the corpus.
id	Short name of corpus (character vector).
home	Path with data directory for indexed corpus.
info	A character vector containing path name of info file.
properties	Named character vector with corpus properties, should at least include 'charset'.
p_attributes	A character vector with positional attributes to declare.
s_attributes	A character vector with structural attributes to declare.
data	A registry_data object.
...	further parameters
property	A single corpus property (character vector).
value	Value of a corpus property (character vector).
info_file	Path to the info file providing information on the corpus.

Details

`registry_file_parse()` will return an object of class `registry_data`.

See the appendix to the 'Corpus Encoding Tutorial' (https://cwb.sourceforge.io/files/CWB_Encoding_Tutorial.pdf), which includes an explanation of the registry file format.

`registry_file_compose` will turn an `registry_data`-object into a character vector with a registry file that can be written to disk.

`registry_file_write()` will compose a registry file from `data` and write it to disk.

`registry_set_property()` will set a single corpus property.

`registry_set_info()` will set the path to the info file.

`registry_set_name()` sets the long descriptive name of the corpus.

Examples

```
regdata <- registry_file_parse(
  corpus = "REUTERS",
  registry_dir = system.file(package = "RcppCWB", "extdata", "cwb", "registry")
  )
```

---

s_attribute_encode *Read, process and write data on structural attributes.*

Description

Read, process and write data on structural attributes.

Usage

```
s_attribute_encode(
  values,
  data_dir,
  s_attribute,
  corpus,
  region_matrix,
  method = c("R", "CWB"),
  registry_dir = Sys.getenv("CORPUS_REGISTRY"),
  encoding,
  delete = FALSE,
  verbose = TRUE
)

s_attribute_recode(
  data_dir,
  s_attribute,
  from = c("UTF-8", "latin1"),
  to = c("UTF-8", "latin1")
```

```
)

s_attribute_files(s_attribute, data_dir)

s_attribute_get_values(s_attribute, data_dir)

s_attribute_get_regions(s_attribute, data_dir)

s_attribute_merge(x, y)

s_attribute_delete(corpus, s_attribute)

s_attribute_rename(corpus, old, new, registry_dir, verbose = TRUE)
```

Arguments

values	A character vector with the values of the structural attribute.
data_dir	The data directory where to write the files.
s_attribute	Name of the structural attribute, an atomic character vector containing only lowercase ASCII characters (a-z), digits (0-9), -, and _: No non-ASCII or uppercase letters allowed.
corpus	A CWB corpus.
region_matrix	A two-column matrix with corpus positions.
method	Either 'R' or 'CWB'.
registry_dir	Path name of the registry directory.
encoding	Encoding of the data.
delete	Logical, whether to call RcppCWB::cl_delete_corpus().
verbose	Logical.
from	Character string describing the current encoding of the attribute.
to	Character string describing the target encoding of the attribute.
x	Data defining a first s-attribute, a data.table (or an object coercible to a data.table) with three columns ("cpos_left", "cpos_right", "value").
y	Data defining a second s-attribute, a data.table (or an object coercible to a data.table) with three columns ("cpos_left", "cpos_right", "value").
old	A character vector with s-attributes to be renamed.
new	A character vector with new names of s-attributes. The vector needs to have the same length as vector old. The 1st, 2nd, 3rd ... nth attribute stated in vector old will get the new names at the 1st, 2nd, 3rd, ... nth position of vector new.

Details

s_attribute_encode() implements a 'pure R' implementation to add or modify structural attributes of an existing CWB corpus.

If the corpus has been loaded/used before, a new s-attribute may not be available unless `RcppCWB::cl_delete_corpus()` has been called. Use the argument `delete` for calling this function.

`s_attribute_recode` will recode the values in the avs-file and change the attribute value index in the avx file. The rng-file remains unchanged. The registry file remains unchanged, and it is highly recommended to consider `s_attribute_recode` as a helper for `corpus_recode` that will recode all s-attributes, all p-attributes, and will reset the encoding in the registry file.

`s_attribute_files()` will return a named character vector with the data files (extensions: "avs", "avx", "rng") in the directory indicated by `data_dir` for the structural attribute `s_attribute`.

`s_attribute_get_values()` is equivalent to performing the CL function cl_struc2id for all strucs of a structural attribute. It is a "pure R" operation that is faster than using CL, as it processes entire files for the s-attribute directly. The return value is a `character` vector with all string values for the s-attribute.

`s_attribute_get_regions` will return a two-column integer matrix with regions for the strucs of a given s-attribute. Left corpus positions are in the first column, right corpus positions in the second column. The result is equivalent to calling RcppCWB::get_region_matrix for all strucs of a s-attribute, but may be somewhat faster. It is a "pure R" function which is fast as it processes files entirely and directly.

`s_attribute_merge()` combines two tables with regions for s-attributes checking for intersections that may cause problems. The heuristic is to keep all non-intersecting annotations and those annotations that define the same region in object `x` and object `y`. Annotations of `x` and `y` which overlap uncleanly, i.e. without an identity of the left and the right corpus position ("cpos_left" / "cpos_right") are dropped. The scenario for using the function is to decode a s-attribute (using `s_attribute_decode()`), mix in an additional annotation, and to re-encode the enhanced s-attribute (using `s_attribute_encode()`).

Function `s_attribute_delete()` is not yet implemented.

Function `s_attribute_rename()` can be used to rename a structural attribute.

See Also

To decode a structural attribute, see `s_attribute_decode`.

Examples

```
require("RcppCWB")
registry_tmp <- fs::path(tempdir(), "cwb", "registry")
data_dir_tmp <- fs::path(tempdir(), "cwb", "indexed_corpora", "reuters")

cwb_dir_rcppcwb <- system.file(package = "RcppCWB", "extdata", "cwb")
registry_dir_rcppcwb <- fs::path(cwb_dir_rcppcwb, "registry")
data_dir_rcppcwb <- fs::path(cwb_dir_rcppcwb,"indexed_corpora", "reuters")

corpus_copy(
  corpus = "REUTERS",
  registry_dir = registry_dir_rcppcwb,
  data_dir = data_dir_rcppcwb,
  registry_dir_new = registry_tmp,
  data_dir_new = data_dir_tmp
)
```

```
no_strucs <- cl_attribute_size(
  corpus = "REUTERS",
  attribute = "id",
  attribute_type = "s",
  registry = registry_tmp
)

cpos_matrix <- get_region_matrix(
      corpus = "REUTERS",
      struc = 0L:(no_strucs - 1L),
      s_attribute = "id",
      registry = registry_tmp
)

s_attribute_encode(
  values = 1L:nrow(cpos_matrix),
  data_dir = data_dir_tmp,
  s_attribute = "article_id",
  corpus = "REUTERS",
  region_matrix = cpos_matrix,
  method = "R",
  registry_dir = registry_tmp,
  encoding = "latin1",
  verbose = TRUE,
  delete = TRUE
)

cl_struc2str(
  "REUTERS",
  struc = 0L:(nrow(cpos_matrix) - 1L),
  s_attribute = "article_id",
  registry = registry_tmp
)

unlink(registry_tmp, recursive = TRUE)
unlink(data_dir_tmp, recursive = TRUE)
data_dir <- system.file(
  package = "RcppCWB",
  "extdata",
  "cwb",
  "indexed_corpora",
  "reuters"
)
avs <- s_attribute_get_values(s_attribute = "id", data_dir = data_dir)
rng <- s_attribute_get_regions(
  s_attribute = "id",
 data_dir = system.file(package = "RcppCWB", "extdata", "cwb", "indexed_corpora", "reuters")
)
x <- data.frame(
  cpos_left =  c(1L, 5L, 10L, 20L, 25L),
  cpos_right = c(2L, 5L, 12L, 21L, 27L),
  value = c("ORG", "LOC", "ORG", "PERS", "ORG"),
```

```
  stringsAsFactors = FALSE
)
y <- data.frame(
  cpos_left =  c(5, 11, 20, 25L, 30L),
  cpos_right = c(5, 12, 22, 27L, 33L),
  value = c("LOC", "ORG", "ORG", "ORG", "ORG"),
  stringsAsFactors = FALSE
)
s_attribute_merge(x,y)
```

`zenodo_get_tarball` *Download corpus tarball from Zenodo*

Description

Download corpus tarball from Zenodo. Downloading both freely available data and data with restricted access is supported.

Usage

```
zenodo_get_tarball(
  url,
  destfile = tempfile(fileext = ".tar.gz"),
  checksum = TRUE,
  verbose = TRUE,
  progress = TRUE
)

gparlsample_url_restricted
```

Arguments

`url`	Landing page at Zenodo for resource. Can also be the URL for restricted access (?token= appended with a long key), or a DOI referencing objects deposited with Zenodo.
`destfile`	A `character` vector with the file path where the downloaded file is to be saved. Tilde-expansion is performed. Defaults to a temporary file.
`checksum`	A `logical` value, whether to check md5 sum.
`verbose`	A `logical` value, whether to output progess messages.
`progress`	A `logical` value, whether to report progress during download.

Format

An object of class `character` of length 1.

Details

A sample subset of the GermaParl corpus is deposited at Zenodo for testing purposes. There are identical open access and restricted versions of GermaParlSample to test different flavours of downloading a resource from Zenodo. The URL for restricted access includes an access token which is very lengthy. This URL is included as a dataset in the package to avoid excessive line in sample code. Note that URLs that give access to restricted data are usually not to be shared.

Value

The filename of the downloaded corpus tarball, designed to serve as input for `corpus_install` (as argument `tarball`). If the resource is not available, `NULL` is returned.

The path of the downloaded resource, or `NULL` if the operation has not been successful.

Examples

```
# Temporary directory structure as a preparatory step
Sys.setenv(CORPUS_REGISTRY = "")
cwb_dirs <- create_cwb_directories(
  prefix = tempdir(),
  ask = FALSE,
  verbose = FALSE
)
Sys.setenv(CORPUS_REGISTRY = cwb_dirs[["registry_dir"]])

# Download and install open access resource
gparl_url_pub <- "https://doi.org/10.5281/zenodo.3823245"
tarball_tmp <- zenodo_get_tarball(url = gparl_url_pub)
if (!is.null(tarball_tmp)) corpus_install(tarball = tarball_tmp)

# Download and install resource with restricted access
tarball_tmp <- zenodo_get_tarball(url = gparlsample_url_restricted)
if (!is.null(tarball_tmp)) corpus_install(tarball = tarball_tmp)
```

Index