Package ‘daiR’

June 11, 2021

Title Interface with Google Cloud Document AI API

Version 0.9.0

Description R interface for the Google Cloud Services 'Document AI API' <https://cloud.google.com/document-ai/> with additional tools for output file parsing and text reconstruction. 'Document AI' is a powerful server-based OCR processor that extracts text and tables from images and pdf files with high accuracy. 'daiR' gives R users programmatic access to this processor and additional tools to handle and visualize the output. See the package website <https://dair.info/> for more information and examples.

Depends R (>= 3.1.0)

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.1.1

Imports base64enc, curl, fs, gargle, glue, googleCloudStorageR, grDevices, httr, jsonlite, magick, pdftools, purrr, stringr

Suggests covr, dplyr, knitr, ngram, qpdf, rmarkdown, sodium, testthat (>= 3.0.0), usethis, utils

VignetteBuilder knitr

URL https://github.com/Hegghammer/daiR, https://dair.info

BugReports https://github.com/Hegghammer/daiR/issues

Config/testthat/edition 3

NeedsCompilation no

Author Thomas Hegghammer [aut, cre] (<https://orcid.org/0000-0001-6253-1518>)

Maintainer Thomas Hegghammer <hegghammer@gmail.com>

Repository CRAN

Date/Publication 2021-06-11 09:20:02 UTC

R topics documented:

`.onAttach` *Run when daiR is attached*

Description

Run when daiR is attached

Usage

```
.onAttach(libname, pkgname)
```

Arguments

`libname`	name of library
`pkgname`	name of package

`build_block_df` *Build block dataframe*

Description

Creates a dataframe with the block bounding boxes identified by Document AI (DAI) in an asynchronous request. Rows are blocks, in the order DAI proposes to read them. Columns are location variables such as page coordinates and page numbers.

Usage

```
build_block_df(json)
```

Arguments

`json`	filepath of a JSON file obtained using `dai_async()`

Details

The location variables are: page number, left boundary, right boundary, top boundary, and bottom boundary.

Value

a block data frame

Examples

```
## Not run:
block_df <- build_block_df("pdf_output.json")

## End(Not run)
```

`build_token_df` *Build token dataframe*

Description

Builds a token dataframe from the text OCRed by Document AI (DAI) in an asynchronous request. Rows are tokens, in the order DAI proposes to read them. Columns are location variables such as page coordinates and block bounding box numbers.

Usage

```
build_token_df(json)
```

Arguments

`json`	filepath of a JSON file obtained using `dai_async()`

Details

The location variables are: start index, end index, left boundary, right boundary, top boundary, bottom boundary, page number, and block number. Start and end indices refer to character position in the string containing the full text.

Value

a token data frame

Examples

```
## Not run:
token_df <- build_token_df("pdf_output.json")

## End(Not run)
```

`dai_async` *OCR documents asynchronously*

Description

Sends files from a Google Cloud Services (GCS) Storage bucket to the GCS Document AI v1 API for asynchronous (offline) processing. The output is delivered to the same bucket as JSON files containing the OCRed text and additional data.

Usage

```
dai_async(
  files,
  dest_folder = NULL,
  bucket = Sys.getenv("GCS_DEFAULT_BUCKET"),
  proj_id = get_project_id(),
  proc_id = Sys.getenv("DAI_PROCESSOR_ID"),
  skip_rev = "true",
  loc = "eu",
  token = dai_token()
)
```

Arguments

`files`	a vector or list of pdf filepaths in a GCS Storage bucket Filepaths must include all parent bucket folder(s) except the bucket name
`dest_folder`	the name of the GCS Storage bucket subfolder where you want the json output
`bucket`	the name of the GCS Storage bucket where the files to be processed are located
`proj_id`	a GCS project id
`proc_id`	a Document AI processor id
`skip_rev`	whether to skip human review; "true" or "false"
`loc`	a two-letter region code; "eu" or "us"
`token`	an access token generated by `dai_auth()` or another auth function

Details

Requires a GCS access token and some configuration of the .Renviron file; see package vignettes for details. Currently, a `dai_async()` call can contain a maximum of 50 files (but a multi-page pdf counts as one file). You can not have more than 5 batch requests and 10,000 pages undergoing processing at any one time. Maximum pdf document length is 2,000 pages. With long pdf documents, Document AI divides the JSON output into separate files ('shards') of 20 pages each. If you want longer shards, use `dai_tab_async()`, which accesses another API endpoint that allows for shards of up to 100 pages.

Value

A list of HTTP responses

Examples

```
## Not run:
# with daiR configured on your system, several parameters are automatically provided,
# and you can pass simple calls, such as:
dai_async("my_document.pdf")

# NB: Include all parent bucket folders (but not the bucket name) in the filepath:
dai_async("for_processing/pdfs/my_document.pdf")

# Bulk process by passing a vector of filepaths in the files argument:
dai_async(my_files)

# Specify a bucket subfolder for the json output:
dai_async(my_files, dest_folder = "processed")

## End(Not run)
```

`dai_async_tab` *OCR asynchronously and get table data*

Description

Sends files from a Google Cloud Services (GCS) Storage bucket to the GCS Document AI v1beta2 API for asynchronous (offline) processing. The output is delivered to the same bucket as JSON files containing the OCRed text and additional information, including table-related data.

Usage

```
dai_async_tab(
  files,
  filetype = "pdf",
  dest_folder = NULL,
  bucket = Sys.getenv("GCS_DEFAULT_BUCKET"),
  proj_id = get_project_id(),
  loc = "eu",
  token = dai_token(),
  pps = 100
)
```

Arguments

`files`	A vector or list of pdf filepaths in a GCS Storage bucket. Filepaths must include all parent bucket folder(s) except the bucket name.
`filetype`	Either "pdf", "gif", or "tiff". If `files` is a vector, all elements must be of the same type.
`dest_folder`	The name of the bucket subfolder where you want the JSON output.
`bucket`	The name of the GCS Storage bucket. Not necessary if you have set a default bucket as a .Renviron variable named `GCS_DEFAULT_BUCKET` as described in the package vignette
`proj_id`	a GCS project id
`loc`	a two-letter region code ("eu" or "us")
`token`	an access token generated by `dai_auth()` or another auth function.
`pps`	an integer from 1 to 100 for the desired number of pages per shard in the JSON output

Details

This function accesses a different API endpoint than the main `dai_async()` function, one that has less language support, but returns table data in addition to parsed text (which `dai_async()` currently does not). This function may be deprecated if/when the v1 API endpoint incorporates table extraction. Use of this service requires a GCS access token and some configuration of the .Renviron file; see vignettes for details. Note that this API endpoint does not require a Document

AI processor id. Maximum pdf document length is 2,000 pages, and the maximum number of pages in active processing is 10,000. Also note that this function does not provide 'true' batch processing; instead it successively submits single requests at 10-second intervals.

Value

A list of HTTP responses

Examples

```
## Not run:
# with daiR configured on your system, several parameters are automatically provided,
# and you can pass simple calls, such as:
dai_async_tab("my_document.pdf")

# NB: Include all parent bucket folders (but not the bucket name) in the filepath:
dai_async_tab("for_processing/pdfs/my_document.pdf")

# Bulk process by passing a vector of filepaths in the files argument:
dai_async_tab(my_files)

# Specify a bucket subfolder for the json output:
dai_async_tab(my_files, dest_folder = "processed")

## End(Not run)
```

`dai_auth` *Check authentication*

Description

Checks whether the user can obtain an access token for Google Cloud Services (GCS) using a service account key stored on file.

Usage

```
dai_auth(
  path = Sys.getenv("GCS_AUTH_FILE"),
  scopes = "https://www.googleapis.com/auth/cloud-platform"
)
```

Arguments

`path`	path to a JSON file with a service account key
`scopes`	GCS auth scopes for the token

Details

daiR takes a very parsimonious approach to authentication, with the native auth functions only supporting service account files. Those who prefer other authentication methods can pass those directly to the `token` parameter in the various functions that call the Document AI API.

Value

no return value, called for side effects

Examples

```
## Not run:
dai_auth()

## End(Not run)
```

`dai_status` *Check job status*

Description

Queries the Google Cloud Services (GCS) Document AI v1 API about the status of a previously submitted asynchronous job.

Usage

```
dai_status(response, loc = "eu", token = dai_token(), verbose = FALSE)
```

Arguments

`response`	a HTTP response object generated by `dai_async()`
`loc`	A two-letter region code; "eu" or "us"
`token`	An authentication token generated by `dai_auth()` or another auth function
`verbose`	Whether to output the full response; boolean

Value

If verbose was set to `TRUE`, a HTTP response object. If verbose was set to `FALSE`, a string summarizing the status.

Examples

```
## Not run:
# Short status message:
response <- dai_async(myfiles)
dai_status(response)

# Full status details:
response <- dai_async(myfiles)
status <- dai_status(response, verbose = TRUE)

## End(Not run)
```

`dai_sync` *OCR document synchronously*

Description

Sends a single document to the Google Cloud Services (GCS) Document AI v1 API for synchronous (immediate) processing. Returns a HTTP response object containing the OCRed text and additional data.

Usage

```
dai_sync(
  file,
  proj_id = get_project_id(),
  proc_id = Sys.getenv("DAI_PROCESSOR_ID"),
  skip_rev = "true",
  loc = "eu",
  token = dai_token()
)
```

Arguments

`file`	path to a single-page pdf or image file
`proj_id`	a GCS project id.
`proc_id`	a Document AI processor id
`skip_rev`	whether to skip human review; "true" or "false".
`loc`	a two-letter region code; "eu" or "us".
`token`	an authentication token generated by `dai_auth()` or another auth function.

Details

Requires a GCS access token and some configuration of the .Renviron file; see package vignettes for details.Input files can be in either .pdf, .bmp, .gif, .jpeg, .jpg, .png, or .tiff format. PDF files can be up to five pages long. Extract the text from the response object with `text_from_dai_response()`. Inspect the entire response object with `httr::content()`.

Value

a HTTP response object

Examples

```
## Not run:
response <- dai_sync("doc_page.pdf")

my_page_scan <- "001.png"
response <- dai_sync(my_page_scan)

## End(Not run)
```

`dai_sync_tab` *OCR synchronously and get table data*

Description

Sends a single document to the Google Cloud Services (GCS) Document AI v1beta2 API for synchronous (immediate) processing. Returns a response object containing the OCRed text and additional information, including table-related data.

Usage

```
dai_sync_tab(file, proj_id = get_project_id(), loc = "eu", token = dai_token())
```

Arguments

`file`	path to a single pdf or image file
`proj_id`	a GCS project id
`loc`	a two-letter region code ("eu" or "us")
`token`	An access token generated by `dai_auth()` or another auth function.

Details

This function accesses a different API endpoint than the main `dai_sync()` function, one that has less language support, but returns table data in addition to parsed text (which `dai_sync()` currently does not). This function may be deprecated if/when the v1 endpoint incorporates table extraction. Use of this service requires a GCS access token and some configuration of the .Renviron file; see vignettes for details. Input files can be in either .pdf, .bmp, .gif, .jpeg, .jpg, .png, or .tiff format. PDFs can be up to five pages long. Extract the text from the response object with `text_from_dai_response()`. Inspect the entire response object with `httr::content()`.

Value

a HTTP response object

Examples

```
## Not run:
response <- dai_sync("doc_page.pdf")

my_page_scan <- "001.png"
response <- dai_sync(my_page_scan)

## End(Not run)
```

dai_token *Produce access token*

Description

Produces an access token for Google Cloud Services (GCS)

Usage

```
dai_token(
  path = Sys.getenv("GCS_AUTH_FILE"),
  scopes = "https://www.googleapis.com/auth/cloud-platform"
)
```

Arguments

`path`	path to a JSON file with a service account key
`scopes`	GCS auth scopes for the token

Value

a GCS access token object (if credentials are valid) or a message (if not).

Examples

```
## Not run:
token <- dai_token()

## End(Not run)
```

dai_user *Get user information*

Description

Fetches the Google Cloud Services (GCS) user information associated with a service account key.

Usage

```
dai_user()
```

Value

a list of user information elements

Examples

```
## Not run:
dai_user()

## End(Not run)
```

draw_blocks *Inspect block bounding boxes*

Description

Plots the block bounding boxes identified by Document AI (DAI) onto images of the submitted document. Generates an annotated .png file for each page in the original document.

Usage

```
draw_blocks(json, dir = getwd())
```

Arguments

`json`	filepath of a JSON file obtained using `dai_async()`
`dir`	path to the desired output directory

Details

Not vectorized, but documents can be multi-page.

Value

no return value, called for side effects

Examples

```
## Not run:
draw_blocks("pdf_output.json", dir = tempdir())

## End(Not run)
```

`draw_lines` *Inspect line bounding boxes*

Description

Plots the line bounding boxes identified by Document AI (DAI) onto images of the submitted document. Generates an annotated .png file for each page in the original document.

Usage

```
draw_lines(json, dir = getwd())
```

Arguments

`json`	filepath of a JSON file obtained using `dai_async()`
`dir`	path to the desired output directory.

Details

Not vectorized, but documents can be multi-page.

Value

no return value, called for side effects

Examples

```
## Not run:
draw_lines("pdf_output.json", dir = tempdir())

## End(Not run)
```

`draw_paragraphs` *Inspect paragraph bounding boxes*

Description

Plots the paragraph bounding boxes identified by Document AI (DAI) onto images of the submitted document. Generates an annotated .png file for each page in the original document.

Usage

```
draw_paragraphs(json, dir = getwd())
```

Arguments

`json`	filepath of a JSON file obtained using `dai_async()`
`dir`	path to the desired output directory.

Details

Not vectorized, but documents can be multi-page.

Value

no return value, called for side effects

Examples

```
## Not run:
draw_paragraphs("pdf_output.json", dir = tempdir())

## End(Not run)
```

`draw_tokens` *Inspect token bounding boxes*

Description

Plots the token (i.e., word) bounding boxes identified by Document AI (DAI) onto images of the submitted document. Generates an annotated .png file for each page in the original document.

Usage

```
draw_tokens(json, dir = getwd())
```

Arguments

`json`	filepath of a JSON file obtained using `dai_async()`
`dir`	path to the desired output directory.

Details

Not vectorized, but documents can be multi-page.

Value

no return value, called for side effects

Examples

```
## Not run:
draw_tokens("pdf_output.json", dir = tempdir())

## End(Not run)
```

`from_labelme` *Extract block coordinates from labelme files*

Description

This is a specialized function for use in connection with text reordering. It takes the output from the image annotation tool 'Labelme' https://github.com/wkentaro/labelme and turns it into a one-row data frame compatible with other 'daiR' functions for text reordering such as `reassign_tokens2()`. See package vignette on text reconstruction for details.

Usage

```
from_labelme(json, page = 1)
```

Arguments

`json`	a json file generated by 'Labelme'
`page`	the number of the annotated page

Value

a data frame with location coordinates for the rectangle marked in 'Labelme'.

Examples

```
## Not run:
new_block <- from_labelme("document1_blocks.json")
new_block <- from_labelme("document5_blocks.json", 5)

## End(Not run)
```

`get_project_id` *Get project id*

Description

Fetches the Google Cloud Services (GCS) project id associated with a service account key.

Usage

```
get_project_id(path = Sys.getenv("GCS_AUTH_FILE"))
```

Arguments

`path`	path to the JSON file with your service account key

Value

a string with a GCS project id

Examples

```
## Not run:
project_id <- get_project_id()

## End(Not run)
```

`image_to_pdf` *Convert images to PDF*

Description

This helper function converts a vector of images to a single PDF.

Usage

```
image_to_pdf(files, pdf_name)
```

Arguments

`files`	a vector of image files
`pdf_name`	a string with the name of the new PDF

Details

Combines any number of image files of almost any type to a single PDF. The vector can consist of different image file types. See the 'Magick' package documentation https://cran.r-project.org/package=magick for details on supported file types. Note that on Linux, ImageMagick may not allow conversion to pdf for security reasons.

Value

no return value, called for side effects

Examples

```
## Not run:
# Single file
new_pdf <- file.path(tempdir(), "document.pdf")
image_to_pdf("document.jpg", new_pdf)

# A vector of image files:
image_to_pdf(images)

## End(Not run)
```

`img_to_binbase` *Image to base64 tiff*

Description

Converts an image file to a base64-encoded binary .tiff file.

Usage

```
img_to_binbase(file)
```

Arguments

`file`	path to an image file

Value

a base64-encoded string

Examples

```
## Not run:
img_encoded <- pdf_to_binbase("image.png")

## End(Not run)
```

`is_json` *Check that a file is JSON*

Description

Checks whether a file is a JSON file.

Usage

```
is_json(file)
```

Arguments

`file`	a filepath

Value

a boolean

Examples

```
## Not run:
is_json("file.json")

## End(Not run)
```

`is_pdf` *Check that a file is PDF*

Description

Checks whether a file is a PDF file.

Usage

```
is_pdf(file)
```

Arguments

`file`	a filepath

Value

a boolean

Examples

```
## Not run:
is_pdf("document.pdf")

## End(Not run)
```

`pdf_to_binbase` *PDF to base64 tiff*

Description

Converts a PDF file to a base64-encoded binary .tiff file.

Usage

```
pdf_to_binbase(file)
```

Arguments

`file`	path to a single-page pdf file

Value

a base64-encoded string

Examples

```
## Not run:
doc_encoded <- pdf_to_binbase("document.pdf")

## End(Not run)
```

`reassign_tokens` *Assign tokens to new blocks*

Description

This is a specialized function for use in connection with text reordering. It modifies a token dataframe by assigning new block bounding box values to a subset of tokens based on prior modifications made to a block dataframe.

Usage

```
reassign_tokens(token_df, block_df)
```

Arguments

`token_df`	a dataframe generated by `build_token_df()`
`block_df`	a dataframe generated by `dair::split_block()` or `dair::build_block_df()`

Details

The token and block data frames provided as input must be from the same JSON output file.

Value

a token data frame

Examples

```
## Not run:
new_token_df <- reassign_tokens(token_df, new_block_df)

## End(Not run)
```

`reassign_tokens2` *Assign tokens to a single new block*

Description

This is a specialized function for use in connection with text reordering. It is designed to facilitate manual splitting of block boundary boxes and typically takes a one-row block dataframe generated by `from_labelme()`.

Usage

```
reassign_tokens2(token_df, block, page = 1)
```

Arguments

`token_df`	a data frame generated by `dair::build_token_df`
`block`	a one-row data frame of the same format as `token_df`
`page`	the number of the page on which the block belongs

Value

a token data frame

Examples

```
## Not run:
new_token_df <- reassign_tokens2(token_df, new_block_df)
new_token_df <- reassign_tokens2(token_df, new_block_df, 5)

## End(Not run)
```

`split_block` *Split a block bounding box*

Description

This function 'splits' (in the sense of changing the coordinates) of an existing block bounding box vertically or horizontally at a specified point. It takes a block data frame as input and modifies it. The splitting produces a new block, which is added to the data frame while the old block's coordinates are updated. The function returns a revised block data frame.

Usage

```
split_block(block_df, page = 1, block, cut_point, direction = "v")
```

Arguments

`block_df`	A dataframe generated by `build_block_df()`.
`page`	The number of the page where the split will be made. Defaults to 1.
`block`	The number of the block to be split.
`cut_point`	A number between 0 and 100, where 0 is the existing left/top limit and 100 is the existing right/bottom limit.
`direction`	"V" for vertical split or "H" for horizontal split. Defaults to "V".

Value

a block data frame

Examples

```
## Not run:
new_block_df <- split_block(df = old_block_df, block = 7, cut_point = 33)

## End(Not run)
```

`tables_from_dai_file` *Get tables from output file*

Description

Extracts all the tables that Document AI (DAI) identified in an asynchronous processing request.

Usage

```
tables_from_dai_file(file)
```

Arguments

`file`	filepath of a JSON file obtained using `dai_async_tab()`

Value

a list of data frames

Examples

```
## Not run:
tables <- tables_from_dai_file("document.json")

## End(Not run)
```

`tables_from_dai_response` *Get tables from response object*

Description

Extracts all the tables that Document AI (DAI) identified in a synchronous processing request.

Usage

```
tables_from_dai_response(object)
```

Arguments

`object`	an HTTP response object returned by `dai_sync_tab()`

Value

a list of data frames

Examples

```
## Not run:
tables <- tables_from_dai_response(response)

## End(Not run)
```

`text_from_dai_file` *Get text from output file*

Description

Extracts the text OCRed by Document AI (DAI) in an asynchronous processing request.

Usage

```
text_from_dai_file(file)
```

Arguments

`file`	filepath of a JSON file obtained using `dai_async()`

Value

a string

Examples

```
## Not run:
text <- text_from_dai_file("output.json")

## End(Not run)
```

`text_from_dai_response` *Get text from HTTP response object*

Description

Extracts the text OCRed by Document AI (DAI) in a synchronous processing request.

Usage

```
text_from_dai_response(response)
```

Arguments

`response`	an HTTP response object returned by `dai_sync()`

Value

a string

Examples

```
## Not run:
text <- text_from_dai_response(response)

## End(Not run)
```

Index