Package 'daiR'

February 12, 2024

Title Interface with Google Cloud Document AI API

Version 1.0.0

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 service that extracts text and tables from images and PDF files with high accuracy. 'daiR' gives R users programmatic access to this service and additional tools to handle and visualize the output. See the package website <https://dair.info/> for more information and examples.

License MIT + file LICENSE

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

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

Depends R (>= 4.2.0)

Imports base64enc, beepr, cli, data.table, fs, gargle, glue, googleCloudStorageR, graphics, grDevices, httr, jsonlite, lifecycle, magick, pdftools, purrr, readtext, stats, stringr, utils, xml2

Suggests knitr, ngram, rmarkdown, testthat (>= 3.1.10)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation no

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

Maintainer Thomas Hegghammer <hegghammer@gmail.com>

Repository CRAN

Date/Publication 2024-02-12 04:20:02 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

Value

no return value, called for side effects

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(object, type = "sync")
```

Arguments

object	either a HTTP response object from `dai_sync()` or the path to a JSON file from `dai_async()`.
type	one of "sync" or "async" depending on the function used to process the original document.

Details

The dataframe variables are: page number, block number, confidence score, left boundary, right boundary, top boundary, and bottom boundary.

Value

a block data frame

Examples

```
## Not run:
resp <- dai_sync("file.pdf")
block_df <- build_block_df(resp)

block_df <- build_block_df("pdf_output.json", type = "async")

## 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(object, type = "sync")
```

Arguments

`object`	either a HTTP response object from `dai_sync()` or the path to a JSON file from `dai_async()`.
`type`	one of "sync" or "async" depending on the function used to process the original document.

Details

The location variables are: token, start index, end index, confidence, 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:
resp <- dai_sync("file.pdf")
token_df <- build_token_df(resp)

token_df <- build_token_df("pdf_output.json", type = "async")

## End(Not run)
```

`create_processor` *Create processor*

Description

Create processor

Usage

```
create_processor(
  name,
  type = "OCR_PROCESSOR",
  proj_id = get_project_id(),
  loc = "eu",
  token = dai_token()
)
```

Arguments

`name`	a string; the proposed display name of the processor.
`type`	a string; one of "OCR_PROCESSOR", "FORM_PARSER_PROCESSOR", "INVOICE_PROCESSOR", or "US_DRIVER_LICENSE_PROCESSOR".
`proj_id`	a GCS project id.
`loc`	a two-letter region code; "eu" or "us".
`token`	an authentication token generated by `dai_auth()` or another auth function.

Details

Creates a Document AI processor and returns the id of the newly created processor. Note that the proposed processor name may already be taken; if so, try again with another name. Consider storing the processor id in an environment variable named DAI_PROCESSOR_ID. For more information about processors, see the Google Document AI documentation at https://cloud.google.com/document-ai/docs/.

Value

a processor id if successful, otherwise NULL.

Examples

```
## Not run:
proc_id <- create_processor("my-processor-123")

## 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"),
  proc_v = NA,
  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
`proc_v`	one of 1) a processor version name, 2) "stable" for the latest processor from the stable channel, or 3) "rc" for the latest processor from the release candidate channel.
`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_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_notify *Notify on job completion*

Description

Queries to the Google Cloud Services (GCS) Document AI API about the status of a previously submitted asynchronous job and emits a sound notification when the job is complete.

Usage

```
dai_notify(response, loc = "eu", token = dai_token(), sound = 2)
```

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
sound	A number from 1 to 10 for the Beepr sound selection (https://www.r-project.org/nosvn/pandoc/beepr.html)

Value

no return value, called for side effects

Examples

```
## Not run:
response <- dai_async(myfiles)
dai_notify(response)

## End(Not run)
```

`dai_status` *Check job status*

Description

Queries the Google Cloud Services (GCS) Document AI 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`	boolean; Whether to output the full response

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"),
  proc_v = NA,
  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.
`proc_v`	one of 1) a processor version name, 2) "stable" for the latest processor from the stable channel, or 3) "rc" for the latest processor from the release candidate channel.
`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")

response <- dai_sync("doc_page.pdf",
                     proc_v = "pretrained-ocr-v1.1-2022-09-12")


## 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)
```

delete_processor *Delete processor*

Description

Delete processor

Usage

```
delete_processor(
  proc_id,
  proj_id = get_project_id(),
  loc = "eu",
  token = dai_token()
)
```

Arguments

proc_id	a Document AI processor id.
proj_id	a GCS project id.
loc	a two-letter region code; "eu" or "us".
token	an authentication token generated by `dai_auth()` or another auth function.

Value

no return value, called for side effects

Examples

```
## Not run:
delete_processor(proc_id = get_processors()$id[1])

## End(Not run)
```

`disable_processor` *Disable processor*

Description

Disable processor

Usage

```
disable_processor(
  proc_id,
  proj_id = get_project_id(),
  loc = "eu",
  token = dai_token()
)
```

Arguments

`proc_id`	a Document AI processor id.
`proj_id`	a GCS project id.
`loc`	a two-letter region code; "eu" or "us".
`token`	an authentication token generated by `dai_auth()` or another auth function.

Value

no return value, called for side effects

Examples

```
## Not run:
disable_processor(proc_id = get_processors()$id[1])

## End(Not run)
```

`draw_blocks` *Draw 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(
  object,
  type = "sync",
  prefix = NULL,
  dir = getwd(),
  linecol = "red",
  linewd = 3,
  fontcol = "blue",
  fontsize = 4
)
```

Arguments

`object`	either a HTTP response object from `dai_sync()` or the path to a JSON file from `dai_async()`.
`type`	one of "sync" or "async", depending on the function used to process the original document.
`prefix`	string to be prepended to the output png filename.
`dir`	path to the desired output directory.
`linecol`	color of the bounding box line.
`linewd`	width of the bounding box line.
`fontcol`	color of the box numbers.
`fontsize`	size of the box numbers.

Details

Not vectorized, but documents can be multi-page.

Value

no return value, called for side effects.

Examples

```
## Not run:
resp <- dai_sync("page.pdf")
draw_blocks(resp)

draw_blocks("page.json", type = "async")

## End(Not run)
```

`draw_entities` *Draw entity bounding boxes*

Description

Plots the entity bounding boxes identified by a Document AI form parser processor onto images of the submitted document. Generates an annotated .png file for each page in the original document.

Usage

```
draw_entities(
  object,
  type = "sync",
  prefix = NULL,
  dir = getwd(),
  linecol = "red",
  linewd = 3,
  fontcol = "blue",
  fontsize = 4
)
```

Arguments

`object`	either a HTTP response object from `dai_sync()` or the path to a JSON file from `dai_async()`.
`type`	one of "sync" or "async", depending on the function used to process the original document.
`prefix`	string to be prepended to the output png filename.
`dir`	path to the desired output directory.
`linecol`	color of the bounding box line.
`linewd`	width of the bounding box line.
`fontcol`	color of the box numbers.
`fontsize`	size of the box numbers.

Details

Not vectorized, but documents can be multi-page.

Value

no return value, called for side effects.

Examples

```
## Not run:
resp <- dai_sync("page.pdf")
draw_entities(resp)

draw_tokens("page.json", type = "async")


## End(Not run)
```

`draw_lines` *Draw 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(
  object,
  type = "sync",
  prefix = NULL,
  dir = getwd(),
  linecol = "red",
  linewd = 3,
  fontcol = "blue",
  fontsize = 4
)
```

Arguments

`object`	either a HTTP response object from `dai_sync()` or the path to a JSON file from `dai_async()`.
`type`	one of "sync" or "async", depending on the function used to process the original document.
`prefix`	string to be prepended to the output png filename.
`dir`	path to the desired output directory.
`linecol`	color of the bounding box line.
`linewd`	width of the bounding box line.
`fontcol`	color of the box numbers.
`fontsize`	size of the box numbers.

Details

Not vectorized, but documents can be multi-page.

Value

no return value, called for side effects.

Examples

```
## Not run:
resp <- dai_sync("page.pdf")
draw_lines(resp)

draw_lines("page.json", type = "async")

## End(Not run)
```

---

draw_paragraphs *Draw 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(
  object,
  type = "sync",
  prefix = NULL,
  dir = getwd(),
  linecol = "red",
  linewd = 3,
  fontcol = "blue",
  fontsize = 4
)
```

Arguments

object	either a HTTP response object from `dai_sync()` or the path to a JSON file from `dai_async()`.
type	one of "sync" or "async", depending on the function used to process the original document.
prefix	string to be prepended to the output png filename.
dir	path to the desired output directory.

linecol	color of the bounding box line.
linewd	width of the bounding box line.
fontcol	color of the box numbers.
fontsize	size of the box numbers.

Details

Not vectorized, but documents can be multi-page.

Value

no return value, called for side effects.

Examples

```
## Not run:
resp <- dai_sync("page.pdf")
draw_paragraphs(resp)

draw_paragraphs("page.json", type = "async")

## End(Not run)
```

---

draw_tokens *Draw 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(
  object,
  type = "sync",
  prefix = NULL,
  dir = getwd(),
  linecol = "red",
  linewd = 3,
  fontcol = "blue",
  fontsize = 4
)
```

Arguments

object	either a HTTP response object from dai_sync() or the path to a JSON file from dai_async().
type	one of "sync" or "async", depending on the function used to process the original document.
prefix	string to be prepended to the output png filename.
dir	path to the desired output directory.
linecol	color of the bounding box line.
linewd	width of the bounding box line.
fontcol	color of the box numbers.
fontsize	size of the box numbers.

Details

Not vectorized, but documents can be multi-page.

Value

no return value, called for side effects.

Examples

```
## Not run:
resp <- dai_sync("page.pdf")
draw_tokens(resp)

draw_tokens("page.json", type = "async")

## End(Not run)
```

---

enable_processor *Enable processor*

Description

Enable processor

Usage

```
enable_processor(
  proc_id,
  proj_id = get_project_id(),
  loc = "eu",
  token = dai_token()
)
```

Arguments

proc_id	a Document AI processor id.
proj_id	a GCS project id.
loc	a two-letter region code; "eu" or "us".
token	an authentication token generated by `dai_auth()` or another auth function.

Value

no return value, called for side effects

Examples

```
## Not run:
enable_processor(proc_id = get_processors()$id[1])

## 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/labelmeai/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_entities *Get entities*

Description

Extracts entities Document AI (DAI) identified by a Document AI form parser processor.

Usage

```
get_entities(object, type = "sync")
```

Arguments

object	either a HTTP response object from dai_sync() or the path to a JSON file from dai_async().
type	one of "sync" or "async", depending on the function used to process the original document.

Value

a list of dataframes, one per page

Examples

```
## Not run:
entities <- get_entities(dai_sync("file.pdf"))

entities <- get_entities("file.json", type = "async")

## End(Not run)
```

get_processors *List created processors*

Description

List created processors

Usage

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

Arguments

proj_id	a GCS project id.
loc	a two-letter region code; "eu" or "us".
token	an authentication token generated by dai_auth() or another auth function.

Details

Retrieves information about the processors that have been created in the current project and are ready for use. For more information about processors, see the Google Document AI documentation at https://cloud.google.com/document-ai/docs/.

Value

a dataframe.

Examples

```
## Not run:
df <- get_processors()

## End(Not run)
```

`get_processor_info` *Get information about processor*

Description

Get information about processor

Usage

```
get_processor_info(
  proc_id,
  proj_id = get_project_id(),
  loc = "eu",
  token = dai_token()
)
```

Arguments

`proc_id`	a Document AI processor id.
`proj_id`	a GCS project id.
`loc`	a two-letter region code; "eu" or "us".
`token`	an authentication token generated by `dai_auth()` or another auth function.

Details

Retrieves information about a processor. For more information about processors, see the Google Document AI documentation at https://cloud.google.com/document-ai/docs/.

Value

a list.

Examples

```
## Not run:
info <- get_processor_info()

info <- get_processor_info(proc_id = get_processors()$id[1])

## End(Not run)
```

`get_processor_versions` *List available versions of processor*

Description

List available versions of processor

Usage

```
get_processor_versions(
  proc_id,
  proj_id = get_project_id(),
  loc = "eu",
  token = dai_token()
)
```

Arguments

`proc_id`	a Document AI processor id.
`proj_id`	a GCS project id.
`loc`	a two-letter region code; "eu" or "us".
`token`	an authentication token generated by `dai_auth()` or another auth function.

Value

a dataframe.

Examples

```
## Not run:
df <- get_processor_versions()

df <- get_processor_versions(proc_id = get_processors()$id[1])

## 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)
```

get_tables *Get tables*

Description

Extracts tables identified by a Document AI form parser processor.

Usage

```
get_tables(object, type = "sync")
```

Arguments

object	either a HTTP response object from `dai_sync()` or the path to a JSON file from `dai_async()`.
type	one of "sync" or "async", depending on the function used to process the original document.

Value

a list of data frames

Examples

```
## Not run:

tables <- get_tables(dai_sync("file.pdf"))

tables <- get_tables("file.json", type = "async")

## End(Not run)
```

`get_text` *Get text*

Description

Extracts the text OCRed by Document AI (DAI)

Usage

```
get_text(
  object,
  type = "sync",
  save_to_file = FALSE,
  dest_dir = getwd(),
  outfile_stem = NULL
)
```

Arguments

`object`	either a HTTP response object from `dai_sync()` or the path to a JSON file from `dai_async()`.
`type`	one of "sync" or "async", depending on the function used to process the original document.
`save_to_file`	boolean; whether to save the text as a .txt file
`dest_dir`	folder path for the .txt output file if `save_to_file = TRUE`
`outfile_stem`	string to form the stem of the .txt output file

Value

a string (if `save_to_file = FALSE`)

Examples

```
## Not run:
text <- get_text(dai_sync("file.pdf"))

text <- get_text("file.json", type = "async", save_to_file = TRUE)

## 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 <- img_to_binbase("image.png")

## End(Not run)
```

is_colour *Check that a string is a valid colour representation*

Description

Checks whether a string is a valid colour representation.

Usage

```
is_colour(x)
```

Arguments

x	a string

Value

a boolean

Examples

```
## Not run:
is_colour("red")
is_colour("#12345")

## 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)
```

---

list_processor_types *List available processor types*

Description

List available processor types

Usage

```
list_processor_types(
  full_list = FALSE,
  proj_id = get_project_id(),
  loc = "eu",
  token = dai_token()
)
```

Arguments

full_list	boolean.
proj_id	a GCS project id.
loc	a two-letter region code; "eu" or "us".
token	an authentication token generated by `dai_auth()` or another auth function.

Details

Retrieves information about the processors that can be created in the current project. With `full_list = TRUE` it returns a list with detailed information about each processor. With `full_list = FALSE` it returns a character vector with just the processor names. For more information about processors, see the Google Document AI documentation at https://cloud.google.com/document-ai/docs/.

Value

list or character vector

Examples

```
## Not run:
avail_short <- list_processor_types()
avail_long <- list_processor_types(full_list = TRUE)

## End(Not run)
```

---

make_hocr *Make hOCR file*

Description

Creates a hOCR file from Document AI output.

Usage

```
make_hocr(type, output, outfile_name = "out.hocr", dir = getwd())
```

Arguments

type	one of "sync" or "async" depending on the function used to process the original document.
output	either a HTTP response object (from dai_sync()) or the path to a JSON file (from dai_async).
outfile_name	a string with the desired filename. Must end with either .hocr, .html, or .xml.
dir	a string with the path to the desired output directory.

Details

hOCR is an open standard of data representation for formatted text obtained from optical character recognition. It can be used to generate searchable PDFs and many other things. This function generates a file compliant with the official hOCR specification (https://github.com/kba/hocr-spec) complete with token-level confidence scores. It also works with non-latin scripts and right-to-left languages.

Value

no return value, called for side effects.

Examples

```
## Not run:
make_hocr(type = "async", output = "output.json")
resp <- dai_sync("file.pdf")
make_hocr(type = "sync", output = resp)
make_hocr(type = "sync", output = resp, outfile_name = "myfile.xml")

## End(Not run)
```

merge_shards *Merge shards*

Description

Merges text files from Document AI output shards into a single text file corresponding to the parent document.

Usage

```
merge_shards(source_dir = getwd(), dest_dir = getwd())
```

Arguments

source_dir	folder path for input files
dest_dir	folder path for output files

Details

The function works on .txt files generated from .json output files, not on .json files directly. It also presupposes that the .txt filenames have the same name stems as the .json files from which they were extracted. For the v1 API, this means files ending with "-0.txt", "-1.txt", "-2.txt", and so forth. The safest approach is to generate .txt files using `get_text()` with the `save_to_file` parameter set to TRUE.

Value

no return value, called for side effects

Examples

```
## Not run:
merge_shards()

merge_shards(tempdir(), getwd())

## 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)
```

redraw_blocks *Inspect revised block bounding boxes*

Description

Tool to visually check the order of block bounding boxes after manual processing (e.g. block reordering or splitting). Takes as its main input a token dataframe generated with `build_token_df()`, `reassign_tokens()`, or `reassign_tokens2()`. The function plots the block bounding boxes onto images of the submitted document. Generates an annotated .png file for each page in the original document.

Usage

```
redraw_blocks(json, token_df, dir = getwd())
```

Arguments

`json`	filepath of a JSON file obtained using `dai_async()`
`token_df`	a token data frame generated with `build_token_df()`, `reassign_tokens()`, or `reassign_tokens2()`.
`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:
redraw_blocks("pdf_output.json", revised_token_df, dir = tempdir())

## 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

[Deprecated] tables_from_dai_file() is deprecated; please use get_text() instead.

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

[Deprecated] tables_from_dai_response() is deprecated; please use get_tables() instead.

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)
```

Index