Package ‘daiR’

September 7, 2023

Title Interface with Google Cloud Document AI API

Version 0.9.9

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.

License MIT + file LICENSE


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

Depends R (>= 4.2.0)

Imports base64enc, beepr, data.table, fs, gargle, glue, googleCloudStorageR, graphics, grDevices, httr, jsonlite, 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 2023-09-07 07:10:13 UTC
R topics documented:

.onAttach ................................................................. 3
build_block_df .......................................................... 3
build_token_df ............................................................ 4
dai_async ................................................................. 5
dai_async_tab ............................................................. 6
dai_auth .................................................................... 8
dai_notify ................................................................. 9
dai_status ................................................................. 9
dai_sync ................................................................. 10
dai_sync_tab .............................................................. 11
dai_token ................................................................. 12
dai_user ................................................................. 13
draw_blocks ............................................................. 13
draw_lines ............................................................... 15
draw_paragraphs .......................................................... 16
draw_tokens ............................................................. 18
from_labelme ............................................................ 19
get_processors .......................................................... 20
get_processor_info ........................................................ 20
get_processor_versions ................................................ 21
get_project_id .......................................................... 22
image_to_pdf ............................................................ 22
img_to_binbase .......................................................... 23
is_colour ................................................................. 24
is_json ................................................................. 24
is_pdf ................................................................. 25
make_hocr ............................................................... 26
merge_shards ............................................................ 27
pdf_to_binbase ........................................................... 28
reassign_tokens .......................................................... 28
reassign_tokens2 .......................................................... 29
redraw_blocks ........................................................... 30
split_block .............................................................. 31
tables_from_dai_file .................................................. 31
tables_from_dai_response ............................................. 32
text_from_dai_file ...................................................... 33
text_from_dai_response ............................................... 33
.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(type, output)

Arguments

Arguments

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

```r
## Not run:
block_df <- build_block_df(type = "async", output = "pdf_output.json")
resp <- dai_sync("file.pdf")
block_df <- build_block_df(type = "sync", output = resp)

## End(Not run)
```

---

### 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

```r
build_token_df(type, output)
```

### 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`).

### 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

```r
## Not run:
token_df <- build_token_df(type = "async", output = "pdf_output.json")
resp <- dai_sync("file.pdf")
token_df <- build_token_df(type = "sync", output = resp)
## 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 OCRd text and additional data.

Usage

```r
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

```r
## 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

```r
dai_async_tab(
  files,
  filetype = "pdf",
  dest_folder = NULL,
  bucket = Sys.getenv("GCS_DEFAULT_BUCKET"),
  proj_id = get_project_id(),
)```
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

```r
## 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)
```
dai_auth

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**

```r
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**

```r
## 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**

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

**Examples**

```r
## Not run:
response <- dai_async(myfiles)
dai_status(response)
## End(Not run)
```
Arguments

response  A HTTP response object generated by `dai_async()` or `dai_tab_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

```r
## 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

```r
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()
)
```
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 OCRred text and additional information, including table-related data.

**Usage**

```r
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

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

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

## End(Not run)
```

---

### `dai_token`

#### Produce access token

### Description

Produces an access token for Google Cloud Services (GCS)

### Usage

```r
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

```r
## 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

```r
## Not run:
dai_user()

## End(Not run)
```

draw_blocks

Draw block bounding boxes on source document

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.
draw_blocks

draw_blocks(
    type,
    output,
    doc = NA,
    prefix = NULL,
    dir = getwd(),
    linecol = "red",
    linewd = 3,
    fontcol = "blue",
    fontsize = 4
)

Arguments

type one of "sync", "async", "sync-tab" or "async-tab", depending on the function used to process the original document.

output either a HTTP response object (from `dai_sync()` or `dai_sync_tab()`) or the path to a JSON file (from `dai_async` or `dai_async_tab()`).

doc filepath to the source document (pdf, tiff, or gif file); only necessary for documents processed with `dai_sync_tab()` or `dai_async_tab()`.

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(type = "sync",
            output = resp)

resp <- dai_sync_tab("page.pdf")
draw_blocks(type = "sync-tab",
            output = resp,
            doc = "page.pdf")
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**

```r
draw_lines(
  type,
  output,
  doc = NA,
  prefix = NULL,
  dir = getwd(),
  linecol = "red",
  linewd = 3,
  fontcol = "blue",
  fontsize = 4
)
```

**Arguments**

- `type` one of "sync", "async", "async-tab" or "async-tab", depending on the function used to process the document.
- `output` either a HTTP response object (from `dai_sync()` or `dai_sync_tab()`) or the path to a JSON file (from `dai_async` or `dai_async_tab()`).
- `doc` filepath to the source document (pdf, tiff, or gif file); only necessary for documents processed with `dai_sync_tab()` or `dai_async_tab()`.
- `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

```r
## Not run:
resp <- dai_sync("page.pdf")
draw_lines(type = "sync",
           output = resp)

resp <- dai_sync_tab("page.pdf")
draw_lines(type = "sync-tab",
           output = resp,
           doc = "page.pdf")

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

draw_lines(type = "async-tab",
           output = "page.json",
           doc = "page.pdf")

## End(Not run)
```

```r

```

## Summary

draw_paragraphs (Draw paragraph bounding boxes on source document)

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(
    type,
    output,
    doc = NA,
    prefix = NULL,
    dir = getwd(),
    linecol = "red",
    linewd = 3,
    fontcol = "blue",
    fontsize = 4
)```
Arguments

- **type**: one of "sync", "async", "sync-tab" or "async-tab", depending on the function used to process the document.
- **output**: either a HTTP response object (from `dai_sync()` or `dai_sync_tab()`), or the path to a JSON file (from `dai_async` or `dai_async_tab()`).
- **doc**: filepath to the source document (pdf, tiff, or gif file); only necessary for documents processed with `dai_sync_tab()` or `dai_async_tab()`.
- **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

```r
## Not run:
resp <- dai_sync("page.pdf")
draw_paragraphs(type = "sync", output = resp)

resp <- dai_sync_tab("page.pdf")
draw_paragraphs(type="sync-tab", output = resp, doc = "page.pdf")

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

draw_paragraphs(type = "async-tab", output = "page.json", doc = "page.pdf")

## 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**

```r
draw_tokens(
  type,  # one of "sync", "async", "sync-tab" or "async-tab", depending on the function used to process the document.
  output,  # either a HTTP response object (from dai_sync() or dai_sync_tab()) or the path to a JSON file (from dai_async or dai_async_tab()).
  doc = NA,  # filepath to the source document (pdf, tiff, or gif file); only necessary for documents processed with dai_sync_tab() or dai_async_tab().
  prefix = NULL,  # string to be prepended to the output png filename.
  dir = getwd(),  # path to the desired output directory.
  linecol = "red",  # color of the bounding box line.
  linewd = 3,  # width of the bounding box line.
  fontcol = "blue",  # color of the box numbers.
  fontsize = 4  # size of the box numbers.
)
```

**Arguments**

- `type` one of "sync", "async", "sync-tab" or "async-tab", depending on the function used to process the document.
- `output` either a HTTP response object (from dai_sync() or dai_sync_tab()) or the path to a JSON file (from dai_async or dai_async_tab()).
- `doc` filepath to the source document (pdf, tiff, or gif file); only necessary for documents processed with dai_sync_tab() or dai_async_tab().
- `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

```r
## Not run:
resp <- dai_sync("page.pdf")
draw_tokens(type = "sync",
            output = resp)

draw_tokens(type = "sync-tab",
            output = resp,
            doc = "page.pdf")

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

draw_tokens(type = "async-tab",
            output = "page.json",
            doc = "page.pdf")

## 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](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

```r
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

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

## End(Not run)
```
get_processors

List all processors available in a project

Description
List all processors available in a project

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
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(
  proj_id = get_project_id(),
  proc_id = Sys.getenv("DAI_PROCESSOR_ID"),
  loc = "eu",
  token = dai_token()
)

get_processor_versions

Arguments

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

Value

a list.

Examples

```r
## Not run:
info <- get_processor_info()
info <- get_processor_info(proc_id = get_processors()$id[1])
## End(Not run)
```

get_processor_versions

*List available processor versions*

Description

List available processor versions

Usage

```r
get_processor_versions(
  proj_id = get_project_id(),
  proc_id = Sys.getenv("DAI_PROCESSOR_ID"),
  loc = "eu",
  token = dai_token()
)
```

Arguments

proj_id  a GCS project id.
proc_id  a Document AI processor 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)

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](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

```r
## 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)
```

---

**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

```r
## Not run:
img_encoded <- pdf_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**

```r
is_colour(x)
```

**Arguments**

- `x`: a string

**Value**

a boolean

**Examples**

```r
## 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**

```r
is_json(file)
```

**Arguments**

- `file`: a filepath
is_pdf

Value

a boolean

Examples

```r
## 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

```r
is_pdf(file)
```

Arguments

- `file`  
a filepath

Value

a boolean

Examples

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

## 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**

```r
## 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

### Description

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

### Usage

```r
merge_shards(source_dir, dest_dir)
```

### 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. For the v1beta2 API, it means files ending with "-page-1-to-100.txt", "-page-101-to-200.txt", etc. The safest approach is to generate .txt files using `text_from_dai_file()` with the `save_to_file` parameter set to TRUE.

### Value

no return value, called for side effects

### Examples

```r
## Not run:
merge_shards(source_dir = getwd(), dest_dir = tempdir())

## End(Not run)
```
pdf_to_binbase  

*PDF to base64 tiff*

**Description**

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

**Usage**

```r
pdf_to_binbase(file)
```

**Arguments**

- `file`  
  path to a single-page pdf file

**Value**

a base64-encoded string

**Examples**

```r
## 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**

```r
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)
**Description**

Tool to visually check the order of block bounding boxes after manual processing (e.g. block re-ordering 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**

```r
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**

```r
## Not run:
redraw_blocks("pdf_output.json", revised_token_df, dir = tempdir())
## End(Not run)
```
split_block

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

```r
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

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

tables_from_dai_file

Description

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

Usage

```r
tables_from_dai_file(file)
```
Arguments

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

Value

a list of data frames

Examples

```r
## 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

```r
## 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**

```r
text_from_dai_file(file, save_to_file = FALSE, dest_dir = getwd())
```

**Arguments**

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

**Value**

a string (if save_to_file = FALSE)

**Examples**

```r
## Not run:
text <- text_from_dai_file("mydoc-0.json")
text_from_dai_file("mydoc-0.json", save_to_file = TRUE)
## 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**

```r
text_from_dai_response(
  response,
  save_to_file = FALSE,
  dest_dir = getwd(),
  filename = "output"
)
```
**Arguments**

- `response` an HTTP response object returned by `dai_sync()`
- `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`
- `filename` string to form the stem of the .txt output file

**Value**

a string (if `save_to_file = FALSE`)

**Examples**

```r
## Not run:

text <- text_from_dai_response(response)

text_from_dai_response(response, save_to_file = TRUE)

## End(Not run)
```
Index

.onAttach, 3
build_block_df, 3
build_token_df, 4
dai_async, 5
dai_async_tab, 6
dai_auth, 8
dai_notify, 9
dai_status, 9
dai_sync, 10
dai_sync_tab, 11
dai_token, 12
dai_user, 13
draw_blocks, 13
draw_lines, 15
draw_paragraphs, 16
draw_tokens, 18
from_labelme, 19
get_processor_info, 20
get_processor_versions, 21
get_processors, 20
get_project_id, 22
image_to_pdf, 22
img_to_binbase, 23
is_colour, 24
is_json, 24
is_pdf, 25
make_hocr, 26
merge_shards, 27
pdf_to_binbase, 28
reassign_tokens, 28
reassign_tokens2, 29
redraw_blocks, 30
split_block, 31
tables_from_dai_file, 31
tables_from_dai_response, 32
text_from_dai_file, 33
text_from_dai_response, 33