Package: openalexConvert (via r-universe)

November 20, 2026

Type Package

Title Convert OpenAlex Parquet (from openalexPro) to other Formats

Version 0.1.3

Description Utilities to convert an OpenAlex parquet/Arrow corpus into other formats. Implemented are at the moment CSL JSON, BibTeX, BibLaTeX, Markdown, LaTeX, HTML, or PDF via Pandoc. Uses DuckDB over Arrow for efficient chunked CSL JSON conversion.

URL https://github.com/openalexPro/openalexConvert, https://openalexpro.github.io/openalexConvert/, https://doi.org/10.5281/zenodo.20448988

BugReports https://github.com/openalexPro/openalexConvert/issues

License MIT + file LICENSE

Depends R (>= 4.1.0)

Imports arrow, DBI, duckdb, httr2, jsonlite, openalexPro (>= 0.11.0), progressr, rmarkdown

Suggests knitr, quarto, testthat (>= 3.0.0)

Additional_repositories https://openalexpro.r-universe.dev

Encoding UTF-8

RoxygenNote 7.3.3

VignetteBuilder quarto

Config/testthat/edition 3

Config/pak/sysreqs cmake make libjq-dev libuv1-dev libxml2-dev libssl-dev

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

Date/Publication 2026-11-20 08:22:09 UTC

RemoteUrl https://github.com/openalexPro/openalexConvert

RemoteRef main

RemoteSha ee92fc31f9f576b79972b0c702805734db0d9cb7

Contents

corpus_export_via_pandoc 2
corpus_to_csljson 3
corpus_to_zotero 4
csljson_convert_pandoc 5
csljson_to_zotero 7
zotero_tags 10

Index **12**

`corpus_export_via_pandoc`

One-shot export via CSL JSON + Pandoc

Description

Convenience wrapper that maps a corpus to CSL JSON, then converts it to the desired output format via Pandoc.

Usage

```
corpus_export_via_pandoc(
  corpus,
  output,
  to = c("bibtex", "biblatex"),
  csl_tmp = NULL,
  ...
)
```

Arguments

`corpus`	Arrow Dataset/Table or data.frame/tibble of works.
`output`	Path to the final file (e.g., ‘corpus.bib‘).
`to`	Target format passed to Pandoc (e.g., ‘"bibtex"‘, ‘"biblatex"‘).
`csl_tmp`	Optional path for a temporary CSL JSON directory. If ‘NULL‘, a temporary directory is used and removed afterwards.
`...`	Additional arguments passed to ‘corpus_to_csljson()‘ (e.g., ‘chunk_size‘).

Value

Invisibly returns the normalized path to the created file.

corpus_to_csljson *Convert a corpus to CSL JSON (chunked)*

Description

Maps an OpenAlex-like corpus (Arrow Dataset/Table or data.frame/tibble) to CSL JSON items and writes them into chunked files. The function creates the directory `output` (if not present) and writes files `chunk_1.json`, `chunk_2.json`, ... inside that directory.

Usage

```
corpus_to_csljson(
  project_dir,
  corpus = file.path(project_dir, "parquet"),
  output = file.path(project_dir, "csljson"),
  chunk_size = 10000,
  overwrite = FALSE,
  verbose = TRUE
)
```

Arguments

`project_dir`	Optional path to project directory. If provided, used to set default values for `corpus` and `output` parameters. Can be omitted if `corpus` and `output` are specified explicitly.
`corpus`	Path to parquet dataset, parquet Dataset/Table (e.g., from `arrow::open_dataset()`) or a data.frame/tibble (e.g., from `dplyr::collect()`).
`output`	Path to a directory to create and populate with chunked CSL JSON files (`chunk_1.json`, `chunk_2.json`, ...).
`chunk_size`	Rows processed per chunk via DuckDB. Default: 10000.
`overwrite`	Overwrite `output` if it exists. Default: FALSE.
`verbose`	Print progress messages. Default: TRUE.

Details

This converter targets the most common OpenAlex field layout and is resilient to missing columns by falling back to `NULL`/empty values in SQL. Mapping includes: title, year, DOI, container-title (venue), volume/issue/pages, authors (with basic given/family split and ORCID when present), URL/abstract, publisher and ISSN, language, keywords (collapsed to a single string), and an `note` whose first line is `OpenAlex: W...` (imported by Zotero into Extra), followed by OA status and citation count. Records are processed in DuckDB-backed chunks for low memory usage.

Value

Invisibly returns `normalizePath(output)`.

corpus_to_zotero *One-shot export of a corpus to Zotero*

Description

Convenience wrapper: maps a corpus to CSL JSON with `corpus_to_csljson()` (in a temporary directory, removed afterwards) and imports it into the library selected in Zotero with `csljson_to_zotero()`. See there for re-exports (`existing`), tags, collections and the import speed.

Usage

```
corpus_to_zotero(
  project_dir,
  corpus = file.path(project_dir, "parquet"),
  existing = c("skip", "update", "overwrite"),
  tags,
  collection,
  chunk_size = 1000,
  ...
)
```

Arguments

`project_dir`	Optional path to the project directory, used for the default of `corpus`.
`corpus`	Path to parquet dataset, Arrow Dataset/Table or data.frame/tibble of works. Default `file.path(project_dir, "parquet")`.
`existing`	`"skip"` (default), `"update"` or `"overwrite"`: what to do with works already in the Zotero library. See `csljson_to_zotero()`.
`tags`	Tags to write, by OpenAlex ID (see "Tags" in `csljson_to_zotero()`): a data.frame, a parquet path, or `NULL` for none. Default: `<project_dir>/zotero_tags.parquet` (written by `zotero_tags()`) if it exists, else `NULL`.
`collection`	Path of the Zotero collection to export into. Default: `c(<name of project_dir>, <name of corpus>)`, leaving out what has no name (a corpus given as a data.frame); `.parquet` is dropped from the corpus name. `NULL` puts the works only into what is selected in Zotero.
`chunk_size`	Works per file sent to Zotero. Default `1000`, which gives progress per 1000 works and stays well within `timeout`.
`...`	Further arguments to `csljson_to_zotero()` (`method`, `url`, `api_key`, `timeout`).

Details

By default the works go into their own nested collection, named after the project directory and the corpus (`<project>` > `<corpus>`), so an accidental export is easy to find and delete. Select the library itself in Zotero (not a collection) before exporting.

Value

Invisibly, the data.frame returned by `csljson_to_zotero()`, one row per chunk.

Examples

```
## Not run:
# In Zotero, select the target library or collection first.
res <- corpus_to_zotero("my_project")
subset(res, !ok)

## End(Not run)
```

`csljson_convert_pandoc`

Convert CSL JSON (file or directory) via Pandoc

Description

Converts CSL JSON with Pandoc into one of: BibTeX, BibLaTeX, Docx, Markdown, LaTeX, or PDF. Behavior depends on `to`:

- `bibtex`/`biblatex`: creates bibliography files. For a directory of chunks, writes `chunk_*.bib` into the directory given by `output`. For a single file, writes the specified `output` (appends `.bib` if missing).
- `docx`/`markdown`/`latex`/`pdf`: renders a formatted references document using citeproc. For a directory of chunks, writes `references.<ext>` inside `output`. For a single file, writes to `output` (appends extension if missing).

Usage

```
csljson_convert_pandoc(
  csljson,
  output,
  to = c("biblatex", "bibtex", "docx", "markdown", "latex", "html", "pdf"),
  from = "csljson",
  overwrite = FALSE,
  verbose = TRUE,
  references_csl = NULL,
  pdf_engine = "xelatex",
  pdf_mainfont = NULL,
  pdf_sansfont = NULL,
  pdf_monofont = NULL,
  pdf_cjk_mainfont = NULL,
  pdf_cjk_options = NULL
)
```

Arguments

csljson	Path to a CSL JSON file (array) or a directory created by corpus_to_csljson() containing chunk_*.json files.
output	Output path. For bib* with a file input, this is the target .bib file (extension added if missing). For bib* with a directory input, this is the output directory. For formatted references (docx, markdown, latex, pdf), this is the output file (file input) or the output directory (dir input; file will be references.<ext> within).
to	One of "biblatex", "bibtex", "docx", "markdown", "latex", "html", or "pdf".
from	Source format; defaults to "csljson".
overwrite	Logical; overwrite existing output file(s). Defaults to FALSE.
verbose	Print progress messages.
references_csl	Optional path to a CSL style file (e.g., apa.csl). If NULL, Pandoc's default style is used.
pdf_engine	LaTeX engine used when to = "pdf". Common values are "xelatex" (default, good Unicode support), "lualatex", or "pdflatex". Passed to Pandoc as --pdf-engine.
pdf_mainfont	Main text font name for PDF output (used with XeLaTeX/LuaLaTeX). Sets Pandoc variable mainfont (e.g., -V mainfont=Source Serif Pro).
pdf_sansfont	Sans-serif font name for PDF output. Sets Pandoc variable sansfont.
pdf_monofont	Monospace font name for PDF output. Sets Pandoc variable monofont.
pdf_cjk_mainfont	Main CJK font name for PDF output. Sets Pandoc variable CJKmainfont for better East-Asian typography.
pdf_cjk_options	Additional CJK options passed via Pandoc variable CJKoptions (e.g., feature flags accepted by xeCJK).

Details

Requires Pandoc to be available. In RStudio, a bundled Pandoc is usually available; otherwise install Pandoc and ensure it is on PATH.

When rendering to = "pdf", this function maps the supplied PDF options to Pandoc command line flags and variables as follows:

- pdf_engine → --pdf-engine=<engine>
- pdf_mainfont, pdf_sansfont, pdf_monofont → -V mainfont=..., -V sansfont=..., -V monofont=...
- pdf_cjk_mainfont, pdf_cjk_options → -V CJKmainfont=..., -V CJKoptions=...

Use these to ensure Unicode coverage and consistent typography, especially for multilingual bibliographies.

Value

Invisibly returns the created file path(s).

`csljson_to_zotero` *Import CSL JSON files into Zotero*

Description

Import one or more CSL JSON files (e.g. the chunks written by `corpus_to_csljson()`) into Zotero.

- `method = "local"` sends the items to a running Zotero desktop app via its local connector endpoint (`/connector/import`, the endpoint the browser connector uses). Zotero's own CSL JSON importer does the mapping. Items go into the library or collection **currently selected in Zotero**; select a group library there to import into the group, and Zotero sync uploads them.
- `method = "web"` (headless upload via the Zotero Web API) is not implemented yet and raises an error.

Zotero imports the CSL `note` into **Extra**; for files written by `corpus_to_csljson()` its first line is the item's `OpenAlex: W...`.

Usage

```
csljson_to_zotero(
  files,
  method = c("local", "web"),
  existing = c("skip", "update", "overwrite"),
  tags = NULL,
  collection = NULL,
  url = "http://127.0.0.1:23119",
  api_key = Sys.getenv("ZOTERO_LOCAL_API_KEY"),
  timeout = 3600
)
```

Arguments

`files`	Character vector. Paths to CSL JSON files. If a single element points to an existing directory, all `*.json` files in that directory (non-recursive) are used.
`method`	`"local"` (default) or `"web"`. See Description.
`existing`	`"skip"` (default), `"update"` or `"overwrite"`: what to do with items whose OpenAlex ID is already in the selected Zotero library. See "Items already in Zotero".
`tags`	`NULL` (default: no tags written), or a data.frame or path to a parquet file with columns `openalex_id`, `tag` and optionally `type` (`0` manual, `1` automatic), such as the one written by `zotero_tags()`. See "Tags".
`collection`	`NULL` (default: only the target selected in Zotero), or the path of a collection to put the works in, e.g. `c("my_project", "corpus_a")`. See "Collection".

`url`	Base URL of the local Zotero server. Default `"http://127.0.0.1:23119"`.
`api_key`	Key for Zotero's local API, only needed for `existing = "update"` or `"overwrite"`, `tags` or `collection`. Defaults to `Sys.getenv("ZOTERO_LOCAL_API_KEY")`; if empty, Zotero is asked for one.
`timeout`	Seconds to wait for Zotero to import one file. Default `3600`; a 10000-item chunk can take 10 to 30 minutes (see "Speed").

Value

Invisibly, a data.frame with one row per file and columns:

file Path to the CSL JSON file.

status_code HTTP status code of the last import request (`201` on success, `NA` if nothing had to be imported).

ok Logical; `TRUE` if every step for the file succeeded.

n_items Number of new items imported.

n_skipped Number of items skipped because they already exist.

n_updated Number of existing Zotero items updated or overwritten.

n_tagged Number of items whose tags were written from `tags`.

message Character; error text if something failed.

Items already in Zotero

Before importing, the selected **library** (all of it, not only the selected collection) is searched for items whose Extra holds an OpenAlex ID, in any of the forms in use: `OpenAlex: W...` (written by this package), `OpenAlex: https://openalex.org/W...` (Zotero's OpenAlex translators), `openalex.work_id: W...` and `OpenAlex Work ID: W...` (two Zotero plugins). Items whose OpenAlex ID is not found there (or that have none) are imported. Items already there are handled by `existing`:

- `"skip"` (default): left untouched and not imported.
- `"update"`: the existing Zotero item gets the new metadata; fields the new metadata does not have keep their old values.
- `"overwrite"`: all fields of the existing Zotero item are replaced by the new metadata; fields it does not have are emptied.

With `"update"` and `"overwrite"` the existing item keeps its item key, child notes and attachments, tags, collections and related items. Technically, the items are imported with Zotero's CSL JSON importer, their fields are copied onto the existing items, and the imported copies are deleted again.

Tags

CSL JSON has no tags, so by default items arrive without tags, and existing items keep theirs. With `tags` (e.g. the `zotero_tags.parquet` written by `zotero_tags()`), tags are written by OpenAlex ID, following `existing`:

- new items get the tags of their OpenAlex ID;

- `"skip"`: existing items are left untouched, tags included;
- `"update"`: tags missing on the existing item are added, its own are kept;
- `"overwrite"`: the existing item's tags are replaced by those in `tags`, and removed if `tags` has none for its OpenAlex ID.

Rows of several libraries are merged; a tag keeps its type (manual or automatic).

Collection

With `collection`, e.g. `c("my_project", "corpus_a")`, the works are also put into that collection, nested from the top level of the selected library (`my_project > corpus_a`). Missing collections are created, existing ones of the same name reused, so a re-export goes into the same collection. Works already in the library join the collection too, with every value of `existing`: collections are independent of the item, and one work can be in several. Other collections of an item are kept.

New items also land in whatever is selected in Zotero, so select the library itself (not a collection) to have them only in `collection`. Deleting the collection in Zotero with its items removes an accidental export again; items that were already in the library before stay in their other collections.

Write access

`"update"`, `"overwrite"`, `tags` and `collection` write through Zotero's local API, which needs Zotero 10 or later and a key. If `api_key` is empty, Zotero shows a dialog asking to allow access for "openalexConvert": choose **Always Allow** (a key allowed once is only good for a single write). The key is kept for the rest of the R session.

Progress is reported via the **progressr** package.

Speed

Zotero gets slower as the library grows. Measured with Zotero 10.0.6 on a Mac, importing 5000 works in files of 1000 into a library starting at about 440 items:

file	items in library before	time	works/s
1	440	57 s	17.5
2	1440	89 s	11.2
3	2440	121 s	8.3
4	3440	145 s	6.9
5	4440	163 s	6.1

Zotero's memory use grew to 3.4 GB during that run. For 100000 works, plan for several hours to more than a day, and consider restarting Zotero every few files.

The search for items already in the library (run once per call, for every value of `existing`) took 0.2 s for 369 IDs and 6.6 s for 5369; it grows faster than the library, so expect tens of minutes for 100000. Re-running all 5000 works with `existing = "skip"` took 4.7 s, mostly that search. `corpus_to_csljson()` itself takes about 1 ms per work.

Examples

```
## Not run:
# In Zotero, select the target library or collection first.
res <- csljson_to_zotero("csljson")  # directory with chunk_1.json, ...
subset(res, !ok)

# Re-export after the corpus was updated: refresh what is already there
csljson_to_zotero("csljson", existing = "update")

## End(Not run)
```

`zotero_tags` *Read Zotero tags by OpenAlex ID*

Description

Reads the tags of all items in the Zotero library **currently selected in Zotero** whose Extra holds an OpenAlex ID (any of the forms listed in `csljson_to_zotero()`) and writes them to a parquet file next to the corpus. Tag works in Zotero, then join the table with the corpus on `openalex_id`.

The whole library is read, not only the selected collection. Rows of other libraries already in `output` are kept; rows of this library are replaced, so the file always holds the library's current tags. Items without an OpenAlex ID in Extra are not included.

Needs the running Zotero desktop app (Zotero 10 or later, for the local API); reading needs no API key. It runs the same search as `csljson_to_zotero()` (see its "Speed" section): a few seconds for a library of 5000 works.

Usage

```
zotero_tags(
  project_dir,
  output = file.path(project_dir, "zotero_tags.parquet"),
  url = "http://127.0.0.1:23119"
)
```

Arguments

`project_dir`	Path to the project directory. Used for the default of `output`.
`output`	Path of the parquet file. Default `file.path(project_dir, "zotero_tags.parquet")`.
`url`	Base URL of the local Zotero server. Default `"http://127.0.0.1:23119"`.

Value

Invisibly, a data.frame with one row per item and tag (the rows of this library), as written to `output`:

openalex_id OpenAlex work ID, e.g. `"W2112574986"`.

tag The tag.

type `0` for a tag added by hand, `1` for one added automatically (e.g. by an import translator).

zotero_library Group ID, or `0` for the personal library.

zotero_key Zotero item key.

A work in the library twice gives rows for each Zotero item.

Examples

```
## Not run:
# In Zotero, select the library first.
tags <- zotero_tags("my_project")
# works tagged "include" in Zotero
unique(subset(tags, tag == "include")$openalex_id)

## End(Not run)
```

Index

corpus_export_via_pandoc, 2
corpus_to_csljson, 3
corpus_to_csljson(), *4*, *7*, *9*
corpus_to_zotero, 4
csljson_convert_pandoc, 5
csljson_to_zotero, 7
csljson_to_zotero(), *4, 5*, *10*

zotero_tags, 10
zotero_tags(), *4*, *7, 8*