Package 'zot'

September 22, 2026

Version 0.1.2

Title Access 'Zotero' Libraries

Description Interface to the 'Zotero' reference manager <https://www.zotero.org> through its public web API and local client database. Provides paginated reads, versioned writes, read-only local queries, batch plans and resumable ledgers, bibliographic comparison, metadata mapping, file attachment, and optional semantic-index queries.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

URL https://averriK.github.io/zot/

Depends R (>= 4.1.0)

Imports data.table, DBI, digest, httr2, jsonlite, RSQLite

Suggests knitr, rmarkdown, testthat (>= 3.2.0)

VignetteBuilder knitr, rmarkdown

SystemRequirements Python 3 and zotero-mcp-server (optional, for ztSemantic())

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Alejandro Verri Kozlowski [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-8535-1170>)

Maintainer Alejandro Verri Kozlowski <averri@fi.uba.ar>

Repository CRAN

Date/Publication 2026-09-22 06:20:02 UTC

Contents

`zotApprove` *Create an approval record for a preview*

Description

Stores a preview digest, optional notes, and a timestamp in an object used by `zotExecute()`.

Usage

```
zotApprove(preview, notes = "")
```

Arguments

`preview`	A `zotPreview()` object.
`notes`	Free-text metadata associated with the record.

Details

This function does not authenticate a person, evaluate an authorization policy, display the plan, or perform a write. The calling application determines how approval records are created. A preview should not be modified after this record is created because `zotExecute()` compares stored digests and does not recalculate the plan digest.

Value

A `zotApproval` containing the preview digest.

See Also

`zotPreview()` and `zotExecute()`.

Examples

```
Plan <- data.table::data.table(
  action = "patch", library = "user", itemKey = "ABCD1234",
  payload = list(list(title = "A reviewed title"))
)
Preview <- zotPreview(Plan, describe = "Synthetic documentation batch")

Approval <- zotApprove(Preview, notes = "Documentation example")
Approval$digest == Preview$digest
```

`zotConfig` *Configure Zotero API access*

Description

Creates a configuration object for requests to the Zotero Web API. The object stores the user identifier, API key, and an optional request performer used by tests or alternative transports. Arguments may be passed explicitly or read from environment variables, which supports interactive, scripted, and agent-driven use.

Usage

```
zotConfig(
  userID = Sys.getenv("ZOTERO_LIBRARY_ID"),
  key = Sys.getenv("ZOTERO_API_KEY"),
  performer = NULL
)
```

Arguments

`userID`	Zotero user identifier. The default reads `ZOTERO_LIBRARY_ID`.
`key`	Zotero API key. The default reads `ZOTERO_API_KEY`.
`performer`	Optional function used by `zotRequest()` instead of the package HTTP transport.

Value

An object of class `zotConfig`.

Examples

```
Config <- zotConfig(userID = "42", key = "example-key")
Config
```

`zotExecute` *Execute a previewed batch with a resumable ledger*

Description

Requires matching approval and preview digests and refuses a ledger written for a different digest. Completed ledger entries are skipped when the same batch resumes. The ledger file is rewritten periodically and at the end.

Usage

```
zotExecute(config, preview, approval, ledgerPath)
```

Arguments

`config`	A `zotConfig()` object.
`preview`	The `zotPreview()` being executed.
`approval`	Its `zotApprove()` record.
`ledgerPath`	JSON ledger file; reuse the same path to resume.

Details

Patch operations read the current item version, use a conditional write, and compare requested fields with a subsequent API read. Create operations record the new key after it can be read. Trash operations use the status returned by `zotTrash()`. The approval record is application metadata; this function does not independently determine who authorized the batch.

Value

The ledger as a list (class `zotLedger`).

See Also

`zotPreview()`, `zotApprove()`, and `zotVerify()`.

Examples

```
Store <- new.env(parent = emptyenv())
Store$version <- 1L
Store$title <- "Before"
Performer <- function(config, method, path, query, body, version) {
  if (method == "GET") {
    return(list(
      status = 200L, headers = list(),
      body = list(version = Store$version, data = list(title = Store$title))
    ))
  }
  Store$title <- body$title
  Store$version <- Store$version + 1L
  list(status = 204L, headers = list(), body = NULL)
}
Config <- structure(
  list(userID = "42", key = "example", apiBase = "https://example.invalid",
       performer = Performer),
  class = "zotConfig"
)
Plan <- data.table::data.table(
  action = "patch", library = "user", itemKey = "K1",
  payload = list(list(title = "After"))
)
Preview <- zotPreview(Plan, describe = "Synthetic offline patch")
Approval <- zotApprove(Preview, notes = "Documentation example")
LedgerPath <- tempfile(fileext = ".json")
Ledger <- zotExecute(Config, Preview, Approval, ledgerPath = LedgerPath)
Ledger$items$K1$status
unlink(LedgerPath)
```

`zotGetAll` *Fetch every page of a listing endpoint*

Description

Follows `Total-Results` with `start`/`limit` pagination and returns the concatenated item list.

Usage

```
zotGetAll(config, path, query = list(), limit = 100L)
```

Arguments

`config`	A `zotConfig()` object.
`path`	API path starting at the host root (see `zotLibraryPath()`).
`query`	Named list of query parameters.
`limit`	Page size.

Value

A list of parsed items.

See Also

zotGet() and zotRequest().

Examples

```
Performer <- function(config, method, path, query, body, version) {
  Page <- if (query$start == 0L) {
    list(list(key = "A"), list(key = "B"))
  } else {
    list(list(key = "C"))
  }
  list(status = 200L, headers = list(`total-results` = "3"), body = Page)
}
Config <- structure(
  list(userID = "42", key = "example", apiBase = "https://example.invalid",
       performer = Performer),
  class = "zotConfig"
)
Items <- zotGetAll(Config, path = "/users/42/items", limit = 2L)
vapply(Items, function(Item) Item$key, character(1L))
```

---

zotLibraryPath *Build a Zotero library API path*

Description

Build a Zotero library API path

Usage

```
zotLibraryPath(config, library = "user")
```

Arguments

config	A zotConfig() object.
library	"user" for the configured user library or a numeric Zotero group identifier.

Value

The API path prefix for the selected library.

See Also

zotConfig() and zotRequest().

Examples

```
Config <- zotConfig(userID = "42", key = "example-key")
zotLibraryPath(Config)
zotLibraryPath(Config, library = "5107760")
```

`zotLocalItems` *List items from the local database*

Description

Returns key, library, type, title, and dates for non-trashed items as a data.table. See `zotLocalQuery()` for synchronization and read-only details.

Usage

```
zotLocalItems(
  library = NULL,
  path = file.path(Sys.getenv("HOME"), "Zotero", "zotero.sqlite")
)
```

Arguments

`library`	`NULL` for every library, `"user"` for the personal library, or a numeric group id.
`path`	Database file.

Value

A data.table with `key`, `groupID` (0 for personal), `itemType`, `title`, `dateAdded` and `dateModified`.

See Also

`zotLocalQuery()`, `ztCensus()`, and `ztSemantic()`.

Examples

```
# A temporary database with the tables used by zotLocalItems().
Path <- tempfile(fileext = ".sqlite")
Connection <- DBI::dbConnect(RSQLite::SQLite(), Path)
DBI::dbWriteTable(Connection, "libraries", data.frame(libraryID = 1:2))
DBI::dbWriteTable(Connection, "groups",
                  data.frame(libraryID = 2L, groupID = 555L))
DBI::dbWriteTable(Connection, "itemTypes",
                  data.frame(itemTypeID = 1L, typeName = "journalArticle"))
DBI::dbWriteTable(Connection, "items", data.frame(
  itemID = 1:3, key = c("AAA", "BBB", "CCC"), libraryID = c(1L, 2L, 1L),
  itemTypeID = 1L, dateAdded = "2026-01-01", dateModified = "2026-01-01"
))
DBI::dbWriteTable(Connection, "fieldsCombined",
```

```
                  data.frame(fieldID = 110L, fieldName = "title"))
DBI::dbWriteTable(Connection, "itemData",
                  data.frame(itemID = 1:3, fieldID = 110L, valueID = 1:3))
DBI::dbWriteTable(Connection, "itemDataValues", data.frame(
  valueID = 1:3, value = c("Personal article", "Group article", "Trashed article")
))
DBI::dbWriteTable(Connection, "deletedItems", data.frame(itemID = 3L))
DBI::dbDisconnect(Connection)

zotLocalItems(library = "user", path = Path)
zotLocalItems(library = 555, path = Path)
unlink(Path)
```

`zotLocalQuery` *Read-only query over the local Zotero database*

Description

Opens the local `zotero.sqlite` database in read-only mode. Local records can lag the Web API until the desktop client synchronizes, and WAL mode can make recent local changes temporarily unavailable to another reader.

Usage

```
zotLocalQuery(
  sql,
  params = list(),
  path = file.path(Sys.getenv("HOME"), "Zotero", "zotero.sqlite")
)
```

Arguments

`sql`	One read-only SQL statement.
`params`	Positional query parameters.
`path`	Database file (default: the standard Zotero location).

Details

The database connection is opened with SQLite's read-only flag and closed when the query finishes. `zotLocalQuery()` does not copy or modify the desktop database. The caller remains responsible for supplying a read-only SQL statement. The default path is `~/Zotero/zotero.sqlite`; pass `path` explicitly for a portable installation or a custom Zotero data directory.

Value

Query result as a data.table.

See Also

zotLocalItems(), zotRequest(), and ztSemantic().

Examples

```
Path <- tempfile(fileext = ".sqlite")
Connection <- DBI::dbConnect(RSQLite::SQLite(), Path)
DBI::dbWriteTable(Connection, "items", data.frame(key = c("A", "B")))
DBI::dbDisconnect(Connection)
zotLocalQuery("SELECT key FROM items ORDER BY key", path = Path)
unlink(Path)
```

---

zotPreview *Preview a batch plan*

Description

Validates and summarizes a plan for execution by zotExecute(). A row has action "patch", "create", or "trash"; a user or group library; an item key for patch and Trash rows; and a payload list.

Usage

```
zotPreview(plan, describe)
```

Arguments

plan	A data.table with columns action, library, itemKey, payload (list column).
describe	Text identifying the batch.

Details

zotPreview() validates the plan, rejects unknown actions and recognized source labels, copies the rows, and calculates a digest from a canonical JSON representation. It does not contact Zotero.

Value

A zotPreview: the plan, its canonical SHA-256 digest, counts and creation time.

See Also

zotApprove(), zotExecute(), and zotVerify().

Examples

```
Plan <- data.table::data.table(
  action = "patch", library = "user", itemKey = "ABCD1234",
  payload = list(list(title = "A reviewed title"))
)
Preview <- zotPreview(Plan, describe = "Correct one title")
Preview
```

`zotRequest` *Perform one Zotero API request*

Description

Core transport for every zot API call: bounded retries, `Backoff` and `Retry-After` honoring, and conditional writes through `If-Unmodified-Since-Version`. Returns for every HTTP status — callers interpret `status` (a 404 is a verdict, not a transport failure); only an exhausted transport raises a `zotHttpError`.

Usage

```
zotRequest(config, method, path, query = list(), body = NULL, version = NULL)

zotGet(config, path, query = list())

zotPost(config, path, body)

zotPatch(config, path, body, version)
```

Arguments

`config`	A `zotConfig()` object.
`method`	HTTP method: `"GET"`, `"POST"` or `"PATCH"`.
`path`	API path starting at the host root (see `zotLibraryPath()`).
`query`	Named list of query parameters.
`body`	Request body, serialized as JSON when not `NULL`.
`version`	Value for `If-Unmodified-Since-Version`; required by `zotPatch()` so no write is unconditional.

Details

A performer receives `config`, `method`, `path`, `query`, `body`, and `version`, and returns a list with `status`, `headers`, and `body`. This makes the complete HTTP boundary replaceable in examples and tests without changing the public request contract. Responses eligible for retry are bounded by five attempts; other HTTP statuses are returned unchanged.

Value

A list with `status`, `headers` (named list) and `body` (parsed JSON, or `NULL` when empty).

See Also

`zotGetAll()`, `zotTrash()`, and `zotPreview()`.

Examples

```
Performer <- function(config, method, path, query, body, version) {
  list(
    status = if (method == "POST") 201L else 200L,
    headers = list(),
    body = list(method = method, path = path, body = body, version = version)
  )
}
Config <- structure(
  list(userID = "42", key = "example", apiBase = "https://example.invalid",
       performer = Performer),
  class = "zotConfig"
)
zotGet(Config, path = "/users/42/items/ABCD1234")$status
zotPost(Config, path = "/users/42/items", body = list(list(title = "New")))$status
zotPatch(Config, path = "/users/42/items/ABCD1234",
         body = list(title = "Revised"), version = 7L)$body$version
```

---

zotSameItem *Strong-identity comparison of two bibliographic records*

Description

A duplicate is only what strong identity proves: an exact DOI, an exact attachment MD5, or normalized title plus year plus first-author family name. A bare title match proves nothing — generic titles collide across distinct documents — and reuse is refused when the first record carries a file the second lacks, whatever the metadata says.

Usage

```
zotSameItem(a, b)
```

Arguments

a, b	Lists with any of `doi`, `md5`, `title`, `year`, `author` (first-author family name) and `hasFile` (logical).

Value

A list: `match` (logical) and `basis` (`"doi"`, `"md5"`, `"title-year-author"` or `"none"`).

See Also

ztTwins() for pairwise candidate detection.

Examples

```
A <- list(doi = "10.1000/Example", hasFile = TRUE)
B <- list(doi = "10.1000/example", hasFile = TRUE)
zotSameItem(A, B)

# A title alone never establishes identity.
zotSameItem(list(title = "Introduction"), list(title = "Introduction"))
```

---

zotTrash *Move items to the recoverable Trash*

Description

The only removal zot implements: `deleted = 1` through a minimal conditional PATCH. Permanent deletion is not part of this package; emptying Trash is not provided by this package.

Usage

```
zotTrash(config, keys, library = "user")
```

Arguments

config	A zotConfig() object.
keys	Character vector of item keys.
library	"user" or a group id (see zotLibraryPath()).

Value

A data.table with one row per key: `itemKey`, `status` ("trashed", "already", "missing" or "failed").

See Also

zotPatch(), zotPreview(), and zotExecute().

Examples

```
Store <- new.env(parent = emptyenv())
Store$version <- 1L
Store$deleted <- 0L
Performer <- function(config, method, path, query, body, version) {
  if (method == "GET") {
    return(list(
      status = 200L,
```

```
      headers = list(),
      body = list(version = Store$version,
                  data = list(deleted = Store$deleted))
    ))
  }
  Store$deleted <- body$deleted
  Store$version <- Store$version + 1L
  list(status = 204L, headers = list(), body = NULL)
}
Config <- structure(
  list(userID = "42", key = "example", apiBase = "https://example.invalid",
       performer = Performer),
  class = "zotConfig"
)
zotTrash(Config, keys = "ABCD1234")
```

`zotVerify` *Census a ledger against its plan*

Description

Summarizes item statuses and re-reads a bounded sample of completed patch rows.

Usage

```
zotVerify(config, preview, ledger)
```

Arguments

`config`	A `zotConfig()` object.
`preview`	The executed `zotPreview()`.
`ledger`	The `zotExecute()` ledger.

Details

At most ten completed patch rows are sampled. Create and Trash rows are not included. The result does not replace inspection of failed or missing ledger entries.

Value

A list: `counts` by status, `sampleChecked`, `sampleConfirmed`.

See Also

`zotExecute()` and `ztCensus()`.

Examples

```
Performer <- function(config, method, path, query, body, version) {
  list(
    status = 200L, headers = list(),
    body = list(version = 2L, data = list(title = "After"))
  )
}
Config <- structure(
  list(userID = "42", key = "example", apiBase = "https://example.invalid",
       performer = Performer),
  class = "zotConfig"
)
Plan <- data.table::data.table(
  action = "patch", library = "user", itemKey = "K1",
  payload = list(list(title = "After"))
)
Preview <- zotPreview(Plan, describe = "Synthetic verified patch")
Ledger <- structure(
  list(items = list(K1 = list(status = "done"))),
  class = "zotLedger"
)
zotVerify(Config, Preview, Ledger)
```

---

ztAttach *Attach a local file to an existing item*

Description

Implements Zotero's full upload protocol: create the attachment item, request upload authorization (an identical file already in storage short-circuits as `exists`), upload to the returned storage URL, register the upload, and verify the attachment's MD5 server-side. Uploads may consume Zotero storage quota and have immediate external effects.

Usage

```
ztAttach(
  config,
  parentKey,
  filePath,
  library = "user",
  filename = basename(filePath),
  contentType = .zotContentType(filePath),
  fileTransport = .zotFileTransport()
)
```

Arguments

`config`	A `zotConfig()` object.

parentKey	Key of the regular item receiving the file, or NA for a standalone attachment.
filePath	Local file to upload.
library	"user" or a group id.
filename	Stored filename; defaults to the file's basename.
contentType	MIME type; defaults by extension for pdf/epub, else application/octet-stream.
fileTransport	The three protocol steps as an injectable list (authorize, upload, register); the default performs them for real. Tests replace it.

Details

The default file transport performs every external protocol step. Supplying fileTransport replaces only authorization, byte upload, and registration; attachment creation and final API verification still use config. A stored file with the same MD5 returns "exists" before uploading bytes.

Value

A list: status ("uploaded", "exists" or "failed"), the attachment key, and error when failed.

See Also

ztCleanName(), zotPreview(), and zotExecute().

Examples

```
Path <- tempfile(fileext = ".pdf")
writeBin(as.raw(1:8), Path)
Performer <- function(config, method, path, query, body, version) {
  list(
    status = 200L, headers = list(),
    body = list(successful = list(`0` = list(key = "ATT1")))
  )
}
Config <- structure(
  list(userID = "42", key = "example", apiBase = "https://example.invalid",
       performer = Performer),
  class = "zotConfig"
)
Transport <- list(
  authorize = function(...) list(exists = 1L),
  upload = function(...) stop("upload should not run"),
  register = function(...) stop("registration should not run")
)
ztAttach(Config, parentKey = NA_character_, filePath = Path,
         fileTransport = Transport)
unlink(Path)
```

ztBlockMoveCreate *Build the create half of a block move*

Description

Produces create rows in the destination library from the given records. Items carrying imported files are refused: moving files needs the upload protocol, which this plan builder does not perform.

Usage

```
ztBlockMoveCreate(records, toLibrary, collectionKey)
```

Arguments

records	A data.table with itemKey, payload (list column of item fields) and hasFile (logical).
toLibrary	Destination library.
collectionKey	Destination collection key.

Value

A guarded-flow plan of create rows, one per record, whose order matches records.

See Also

ztBlockMoveTrash(), zotPreview(), and ztAttach().

Examples

```
Records <- data.table::data.table(
  itemKey = "OLD1",
  payload = list(list(itemType = "book", title = "Reviewed book")),
  hasFile = FALSE
)
ztBlockMoveCreate(Records, toLibrary = "5107760", collectionKey = "DEST")
```

ztBlockMoveTrash *Build the trash half of a block move from its executed create ledger*

Description

Returns Trash rows for originals whose corresponding create-ledger entry is marked "done" and records a new item key.

Usage

```
ztBlockMoveTrash(records, createLedger, fromLibrary)
```

Arguments

records	The records given to ztBlockMoveCreate().
createLedger	The zotExecute() ledger of the create plan.
fromLibrary	Library holding the originals.

Value

A guarded-flow plan of trash rows.

See Also

ztBlockMoveCreate(), zotExecute(), and zotTrash().

Examples

```
Records <- data.table::data.table(
  itemKey = "OLD1", payload = list(list(title = "Reviewed book")),
  hasFile = FALSE
)
Ledger <- list(items = list(`create-1` = list(status = "done", new = "NEW1")))
ztBlockMoveTrash(Records, createLedger = Ledger, fromLibrary = "user")
```

---

ztCensus *Census a library through the API*

Description

Returns top-level and Trash counts for one library, plus per-collection counts when collection keys are supplied.

Usage

```
ztCensus(config, library = "user", collectionKeys = NULL)
```

Arguments

config	A zotConfig() object.
library	"user" or a group id.
collectionKeys	Optional named character vector: name -> key.

Details

Each scope is counted from the API Total-Results response header while requesting only one record. Supplying named collection keys adds one row per requested collection without downloading the library contents.

Value

A data.table with `scope` and `total`.

See Also

`zotGet()`, `zotLocalItems()`, and `zotVerify()`.

Examples

```
Performer <- function(config, method, path, query, body, version) {
  Total <- if (grepl("/trash", path)) "7" else "100"
  list(status = 200L, headers = list(`total-results` = Total), body = list())
}
Config <- structure(
  list(userID = "42", key = "example", apiBase = "https://example.invalid",
       performer = Performer),
  class = "zotConfig"
)
ztCensus(Config)
```

---

ztCleanName *Normalize an incoming filename*

Description

Removes source labels recognized by the package and copy markers such as `(1)` before the extension. The extension and other parenthetical text are retained.

Usage

```
ztCleanName(name)
```

Arguments

`name`	A filename.

Value

The cleaned filename.

See Also

`ztCleanTitle()` and `ztCleanPayload()`.

Examples

```
ztCleanName("Dynamics (1).pdf")
ztCleanName("Course Notes (SEG Course).pdf")
```

ztCleanPayload *Sanitize every text field of an incoming item payload*

Description

Applies the package's text-normalization rules across an item payload. Titles lose trailing extensions and copy markers, filenames retain their extensions, and creator names and tags are included. Non-text values are returned unchanged.

Usage

```
ztCleanPayload(payload)
```

Arguments

payload	An item payload (named list of fields).

Value

The payload with every text field cleaned.

See Also

ztCleanName(), ztCleanTitle(), and zotPreview().

Examples

```
Payload <- list(
  itemType = "book",
  title = "Structural Analysis (1).pdf",
  creators = list(list(creatorType = "author", firstName = "A.",
                       lastName = "Author")),
  filename = "Structural Analysis (1).pdf"
)
ztCleanPayload(Payload)
```

ztCleanTitle *Normalize an incoming title*

Description

Like ztCleanName(), and additionally strips a trailing file extension and trailing copy markers — a title is not a filename.

Usage

```
ztCleanTitle(title)
```

Arguments

`title`	An item title.

Value

The cleaned title.

See Also

`ztCleanName()`, `ztCleanPayload()`, and `ztRetitle()`.

Examples

```
ztCleanTitle("Dynamics of Structures (1).pdf")
ztCleanTitle("Optimization (2nd Edition)")
```

---

ztCollectionTree *Ensure a collection tree exists in a library*

Description

Find or create each path (vectors of names, root first) and return the full path to key map. Creating a missing collection has an immediate API effect.

Usage

```
ztCollectionTree(config, library, paths)
```

Arguments

`config`	A `zotConfig()` object.
`library`	`"user"` or a group id.
`paths`	A list of character vectors, e.g. `list(c("Conferences", "ICOLD11"))`.

Details

Existing collections are indexed by parent key and name. Missing nodes are created from the root downward and reused by later paths in the same call. Unlike the plan builders, this function has immediate external effects.

Value

Named character vector: `"A/B"` -> collection key.

See Also

`ztRefile()` and `zotPreview()`.

Examples

```
Counter <- new.env(parent = emptyenv())
Counter$n <- 0L
Performer <- function(config, method, path, query, body, version) {
  if (method == "GET") {
    return(list(
      status = 200L, headers = list(`total-results` = "0"), body = list()
    ))
  }
  Counter$n <- Counter$n + 1L
  Key <- paste0("COLL", Counter$n)
  list(
    status = 200L, headers = list(),
    body = list(successful = list(`0` = list(key = Key)))
  )
}
Config <- structure(
  list(userID = "42", key = "example", apiBase = "https://example.invalid",
       performer = Performer),
  class = "zotConfig"
)
ztCollectionTree(Config, library = "user", paths = list(c("Projects", "2026")))
```

---

ztCrossref *Map a Crossref record to a Zotero item payload*

Description

The returned payload is cleaned with `ztCleanPayload()` and can be placed in a create plan. The default fetcher contacts the public Crossref API; inject a compatible function for reproducible or offline work.

Usage

```
ztCrossref(doi, getJson = .zotCrossrefGet)
```

Arguments

doi	The DOI to resolve.
getJson	Injectable fetcher returning the parsed Crossref message for a DOI; the default performs the real request.

Value

An item payload for a create row, or a classed error when the DOI does not resolve.

See Also

`ztCleanPayload()` and `zotPreview()`.

Examples

```
Message <- function(doi) list(
  title = list("A reproducible record"),
  author = list(list(given = "Ada", family = "Lovelace")),
  `container-title` = list("Example Journal"),
  issued = list(`date-parts` = list(list(2026L)))
)
ztCrossref("10.1000/example", getJson = Message)
```

`ztRefile` *Build a refile plan that adds items to a collection*

Description

Fetches each item's current memberships at plan time and produces patch rows whose `collections` payload is the union with `addTo` — the Zotero API replaces the whole membership list, so the merge is mandatory.

Usage

```
ztRefile(config, itemKeys, library, addTo)
```

Arguments

`config`	A `zotConfig()` object.
`itemKeys`	Character vector of item keys.
`library`	`"user"` or a group id.
`addTo`	Collection key to add.

Details

Planning performs one current API read per key. It does not write, and it omits missing items and items already in the destination collection. Memberships can change after planning; rebuild or review the plan before execution when concurrent changes are possible.

Value

A guarded-flow plan (items already members are omitted).

See Also

`ztCollectionTree()`, `zotPreview()`, and `zotExecute()`.

Examples

```
Memberships <- list(K1 = list("OLD"), K2 = list("TARGET"))
Performer <- function(config, method, path, query, body, version) {
  Key <- sub(".*/items/", "", path)
  list(
    status = 200L, headers = list(),
    body = list(version = 1L, data = list(collections = Memberships[[Key]]))
  )
}
Config <- structure(
  list(userID = "42", key = "example", apiBase = "https://example.invalid",
       performer = Performer),
  class = "zotConfig"
)
ztRefile(Config, itemKeys = c("K1", "K2"), library = "user",
         addTo = "TARGET")
```

`ztRetitle` *Build a retitle plan from a deterministic transformation*

Description

Pure plan builder: applies `transform` to every title and returns patch rows only where the result differs. Feed the result to `zotPreview()`.

Usage

```
ztRetitle(items, transform)
```

Arguments

`items`	A data.table with `itemKey`, `library`, `title`.
`transform`	A function from one title to its cleaned form.

Value

A guarded-flow plan (possibly empty).

See Also

`ztCleanTitle()` and `zotPreview()`.

Examples

```
Items <- data.table::data.table(
  itemKey = c("A", "B"), library = "user",
  title = c("Reviewed book (1).pdf", "Existing title")
)
ztRetitle(Items, transform = ztCleanTitle)
```

`ztSemantic` *Query the semantic index built by zotero-mcp*

Description

Read-only query over the chroma embedding index that the companion zotero-mcp tool builds (`zotero-mcp update-db`). Building or refreshing that index is a separate operation of the companion tool and is never triggered by this function. Requires the companion tool's Python environment; results can lag current Zotero API data.

Usage

```
ztSemantic(
  query,
  limit = 10L,
  dbPath = file.path(Sys.getenv("HOME"), ".config", "zotero-mcp", "chroma_db"),
  pythonPath = file.path(Sys.getenv("HOME"), ".local", "share", "uv", "tools",
    "zotero-mcp-server", "bin", "python3")
)
```

Arguments

`query`	Natural-language query text.
`limit`	Maximum results.
`dbPath`	Chroma database directory.
`pythonPath`	Python interpreter of the companion environment.

Details

The defaults target one common Unix `uv` tool installation and index layout; pass `pythonPath` and `dbPath` for other environments. The function requires local access to a compatible Chroma collection named `zotero_library`, launches that Python interpreter once, and converts its JSON result to a data.table. It never creates or refreshes the semantic index. The current query has no library selector: `limit` applies across all indexed personal and group records, whose `groupID` is returned for inspection.

Value

A data.table with `itemKey`, `groupID`, `title`, `distance` (smaller is closer).

See Also

`zotLocalItems()` and `zotLocalQuery()`.

Examples

```
# Requires the companion Python environment and a populated semantic index.
# These paths match the default Unix uv installation; adapt them as needed.
Python <- file.path(path.expand("~"), ".local", "share", "uv", "tools",
                    "zotero-mcp-server", "bin", "python3")
Index <- file.path(path.expand("~"), ".config", "zotero-mcp", "chroma_db")
if (file.exists(Python) && file.exists(file.path(Index, "chroma.sqlite3"))) {
  ztSemantic("conditional mean spectrum", limit = 5L,
             pythonPath = Python, dbPath = Index)
}
```

`ztTwins` *Detect duplicate candidates with evidence classes*

Description

Groups records by normalized title to nominate candidates, then judges every pair with `zotSameItem()` — the strong-identity rule decides, never the title alone.

Usage

```
ztTwins(records)
```

Arguments

`records`	A data.table with `itemKey` and any of `title`, `year`, `author`, `doi`, `md5`, `hasFile`.

Value

A data.table of pairs: `keyA`, `keyB`, `match`, `basis`.

See Also

`zotSameItem()` and `zotPreview()`.

Examples

```
Records <- data.table::data.table(
  itemKey = c("A", "B"), title = c("Paper", "Paper"),
  year = c(2026L, 2026L), author = c("Lovelace", "Lovelace"),
  doi = NA_character_, md5 = NA_character_, hasFile = FALSE
)
ztTwins(Records)
```

Index