

# Package ‘okf’

June 30, 2026

**Title** Open Knowledge Format (OKF) Ingestion

**Version** 0.5.2

**Description** Read, validate, and load Open Knowledge Format (OKF) bundles (a directory of markdown files with YAML frontmatter) into a portable DuckDB catalog, build the concept graph, render to HTML, and optionally embed concept bodies for semantic search. Deterministic and agent-free: the same bundle always yields the same catalog, graph, and render, with no LLM calls in the core. Conformant and permissive per the OKF v0.1 specification.

**License** Apache License (>= 2)

**Encoding** UTF-8

**Depends** R (>= 4.1.0)

**Imports** yaml, DBI, duckdb, digest, jsonlite, utils

**Suggests** httr2, commonmark, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**RoxygenNote** 7.3.2

**NeedsCompilation** no

**Author** Travis Jakel [aut, cre]

**Maintainer** Travis Jakel <travis.s.jakel@gmail.com>

**Repository** CRAN

**Date/Publication** 2026-06-30 10:40:07 UTC

## Contents

okf_backlinks . . . . .	2
okf_chunk_body . . . . .	3
okf_clusters . . . . .	3
okf_context . . . . .	4
okf_doctor . . . . .	4
okf_doctor_fix . . . . .	5
okf_embed . . . . .	6
okf_extract_links . . . . .	6

okf_fetch . . . . .	7
okf_graph_html . . . . .	7
okf_graph_json . . . . .	8
okf_graph_mermaid . . . . .	8
okf_html . . . . .	9
okf_impact . . . . .	9
okf_ingest . . . . .	10
okf_links . . . . .	11
okf_ollama_embedder . . . . .	11
okf_parse_file . . . . .	12
okf_query . . . . .	12
okf_rag . . . . .	13
okf_read . . . . .	13
okf_resolve_link . . . . .	14
okf_validate . . . . .	14

<b>Index</b>	<b>15</b>
--------------	-----------

---

okf_backlinks	<i>Concepts that link to a given concept ("linked from" / backlinks).</i>
---------------	---

---

## Description

Concepts that link to a given concept ("linked from" / backlinks).

## Usage

```
okf_backlinks(con, path)
```

## Arguments

con	An open DuckDB connection to an okf catalog.
path	Bundle-relative concept path.

## Value

Character vector of source concept paths (resolved inbound links).

---

okf_chunk_body	<i>Split a concept body into chunks on paragraph boundaries.</i>
----------------	--

---

**Description**

Split a concept body into chunks on paragraph boundaries.

**Usage**

```
okf_chunk_body(body, target_chars = 600L)
```

**Arguments**

body	Concept body text.
target_chars	Approximate maximum chunk size in characters.

**Value**

Character vector of chunks.

---

okf_clusters	<i>Deterministic community labels via synchronous label propagation.</i>
--------------	--

---

**Description**

Operates on the undirected resolved-link graph. Each node starts in its own community; nodes iteratively adopt the most common label among neighbours, ties broken by the lexicographically smallest label (so the result is fully reproducible – no randomness). Isolated nodes keep their own label.

**Usage**

```
okf_clusters(con, max_iter = 50L, include_reserved = FALSE)
```

**Arguments**

con	An open DuckDB connection to an okf catalog.
max_iter	Maximum propagation sweeps.
include_reserved	Include reserved concepts ('index.md'/'log.md') as nodes – useful for graph visualization, where 'index.md' is the hub.

**Value**

A data.frame with 'path' and integer 'cluster' (1-based, stable order).

---

okf_context	<i>Assemble an index-first, link-following slice of a bundle as one markdown blob for direct LLM consumption.</i>
-------------	---

---

### Description

This is the OKF / "LLM wiki" consume primitive (Karpathy): hand the agent 'index.md' plus the relevant concept(s) and their link-neighborhood to read directly. It uses the concept graph – **\*\*no embeddings, no vector search\*\***. With 'start', it walks the (undirected) link graph from that concept to 'depth'; without 'start', it packs all concepts. Output is capped to roughly 'max\_tokens' (estimated at ~4 chars/token).

### Usage

```
okf_context(
  con,
  start = NULL,
  depth = 1L,
  max_tokens = 8000L,
  include_index = TRUE
)
```

### Arguments

con	An open DuckDB connection to an okf catalog.
start	Optional concept path to center the neighborhood on.
depth	Link-graph radius around 'start' (ignored when 'start' is NULL).
max_tokens	Approximate output budget.
include_index	Prepend 'index.md' (the map) when present.

### Value

A list with 'text' (the markdown blob), 'included'/'omitted' concept paths, and 'est\_tokens'.

---

okf_doctor	<i>Health / maintenance report for an ingested OKF catalog.</i>
------------	---

---

### Description

Combines the validation findings already stored in the catalog (missing type, broken links, orphans, non-ISO timestamps, ...) with maintenance checks (duplicate titles; and, when 'now' is supplied, future/stale timestamps), and computes a health 'score' = the percentage of non-reserved concepts with zero findings. Fully deterministic.

**Usage**

```
okf_doctor(con, now = NULL, stale_days = NULL)
```

**Arguments**

con	An open DuckDB connection to an okf catalog.
now	Optional ISO-8601 "current time" enabling stale/future-timestamp checks (kept explicit so the function stays deterministic; the CLI passes the wall clock).
stale_days	Optional integer; with 'now', flag timestamps older than this many days.

**Value**

A list with 'score', 'n\_concepts', 'n\_healthy', 'n\_error', 'n\_warn', 'by\_rule' (named counts), and 'issues' (a data.frame of path/severity/rule/ message).

---

okf_doctor_fix	<i>Apply only unambiguously-safe maintenance fixes to a bundle's source files.</i>
----------------	--

---

**Description**

Two mechanical, deterministic repairs (never invents content):

- **timestamps** – a parseable non-ISO 'timestamp:' is rewritten to ISO-8601.
- **moved links** – a broken link whose basename matches *exactly one* concept is re-pointed to that concept (relative to the linking file).

Edits files in place. Anything ambiguous is left for [okf\_doctor()] to report.

**Usage**

```
okf_doctor_fix(root)
```

**Arguments**

root	A bundle directory path.
------	--------------------------

**Value**

A data.frame of changes ('path', 'kind', 'before', 'after'); zero rows if nothing was safely fixable.

---

okf_embed	<i>Chunk and embed concept bodies into the catalog for semantic search.</i>
-----------	---

---

**Description**

Populates 'okf\_chunk' with one row per chunk plus its embedding vector and the concept's 'content\_hash'. By default replaces all chunks. With 'incremental = TRUE', only concepts whose 'content\_hash' differs from what was last embedded are re-embedded (and removed concepts' chunks are dropped) – the expensive embedder calls are skipped for unchanged concepts.

**Usage**

```
okf_embed(con, embedder = NULL, target_chars = 600L, incremental = FALSE)
```

**Arguments**

con	An open DuckDB connection to an okf catalog.
embedder	An embedder function; defaults to [okf_ollama_embedder()].
target_chars	Approximate chunk size in characters.
incremental	Re-embed only concepts whose content changed.

**Value**

The number of chunks (re)written this call (invisibly usable as an integer).

---

okf_extract_links	<i>Extract markdown link targets from a concept body (OKF cross-links, sec. 4).</i>
-------------------	---

---

**Description**

Extract markdown link targets from a concept body (OKF cross-links, sec. 4).

**Usage**

```
okf_extract_links(body)
```

**Arguments**

body	Concept body text.
------	--------------------

**Value**

Character vector of raw link targets (as written).

---

okf_fetch	<i>Materialize an OKF bundle from a directory, git URL, or tar/zip archive.</i>
-----------	---

---

### Description

Local directories are used in place. Git URLs (github/gitlab/bitbucket, ‘.git’, or ‘git@’) are shallow-cloned. Tar/zip archives (local path or ‘http(s)’ URL) are downloaded if remote and extracted. The caller **MUST** invoke the returned ‘cleanup()’ when done to remove any temporary files.

### Usage

```
okf_fetch(source, subdir = NULL, branch = NULL)
```

### Arguments

source	A directory path, git URL, or tar/zip path/URL.
subdir	Optional bundle path within the cloned/extracted tree.
branch	Optional git branch or tag (git sources only).

### Value

A list with ‘dir’ (the resolved bundle directory), ‘source\_kind’ (“dir”/“git”/“tar”/“zip”), and ‘cleanup’ (a function).

---

okf_graph_html	<i>Render the concept graph as one self-contained interactive HTML page.</i>
----------------	--

---

### Description

A force-directed graph drawn on a ‘<canvas>’ with hand-rolled vanilla JS (no CDN, no framework) – pan, zoom, drag, type-to-search, nodes coloured by community ([okf\_clusters()]). Clicking a node navigates to its rendered ‘.html’ (relative), so dropping ‘graph.html’ into an [okf\_html()] site root makes the graph a live map of the site. Fully offline; embeds the node/edge model as JSON.

### Usage

```
okf_graph_html(con, out, site_title = NULL)
```

### Arguments

con	An open DuckDB connection to an okf catalog.
out	Output ‘.html’ file path.
site_title	Optional page title; defaults to the bundle directory name.

**Value**

The output path (invisibly).

---

okf_graph_json	<i>Export the concept graph as portable JSON (nodes and edges).</i>
----------------	---

---

**Description**

Returns a JSON object with ‘nodes’ and ‘edges’. Nodes carry ‘id’ (path), ‘type’, ‘title’, ‘tags’, ‘cluster’ (from [okf\_clusters()]), and ‘href’ (the rendered ‘.html’ path). Edges are resolved links with ‘source’ and ‘target’ fields. Feeds any external graph visualizer – the same “core is a contract” idea as the DuckDB catalog.

**Usage**

```
okf_graph_json(con, pretty = TRUE)
```

**Arguments**

con	An open DuckDB connection to an okf catalog.
pretty	Pretty-print the JSON.

**Value**

A JSON string (invisibly also suitable for writing to a file).

---

okf_graph_mermaid	<i>Render the concept graph as a Mermaid ‘graph LR’ diagram.</i>
-------------------	--

---

**Description**

A text diagram for embedding directly in markdown (READMEs, docs, GitHub renders it natively) – the lightweight complement to the interactive [okf\_graph\_html()]. Node ids are sanitized; labels are the concept titles.

**Usage**

```
okf_graph_mermaid(con)
```

**Arguments**

con	An open DuckDB connection to an okf catalog.
-----	--

**Value**

A Mermaid diagram as a single string (a “““mermaid””” block).

---

okf_html	<i>Render an ingested OKF catalog to HTML for viewing.</i>
----------	--

---

### Description

Two modes. As a navigable **site** (`single = FALSE`, the default), writes one self-contained `.html` per concept under `out/` (mirroring the bundle's directory tree) plus an `index.html` landing page; internal `.md` links are rewritten to `.html`. As a **single file** (`single = TRUE`), writes one self-contained `.html` at path `out`, with each concept an anchored `<section>` and intra-bundle links rewritten to in-page anchors. No JavaScript; CSS is inlined so output is portable. Reserved concepts (`index.md`, `log.md`) are rendered too. Bodies are rendered with the commonmark package; broken/orphan links are surfaced in a per-page footer badge from the validation findings.

### Usage

```
okf_html(con, out, single = FALSE, site_title = NULL)
```

### Arguments

<code>con</code>	An open DuckDB connection to an okf catalog (from <code>[okf_ingest()]</code> ).
<code>out</code>	Output directory (site mode) or output <code>.html</code> file path (single).
<code>single</code>	Emit one self-contained file instead of a per-concept site.
<code>site_title</code>	Optional title for the landing page / single-file header; defaults to the bundle directory name.

### Value

A list with `'files'` (paths written), `'n_concepts'`, and `'mode'` (invisibly).

---

okf_impact	<i>Link-impact ("ripple") of a concept.</i>
------------	---

---

### Description

Reports direct `'outbound'` (concepts it links to), direct `'inbound'` (concepts linking to it, i.e. back-links), and `'transitive'` – every concept that can reach it by following resolved links (what a change here could ripple to).

### Usage

```
okf_impact(con, path)
```

### Arguments

<code>con</code>	An open DuckDB connection to an okf catalog.
<code>path</code>	Bundle-relative concept path.

**Value**

A list with 'path', 'outbound', 'inbound', 'transitive' (all sorted character vectors).

---

okf\_ingest

*Ingest an OKF bundle into a DuckDB catalog.*

---

**Description**

Reads, validates, and loads the bundle into the 'okf\_bundle', 'okf\_concept', 'okf\_link', and 'okf\_validation' tables of a (file or in-memory) DuckDB database.

**Usage**

```
okf_ingest(
  root,
  db_path = ":memory:",
  ingested_at = NULL,
  bundle_id = NULL,
  source_kind = "dir",
  subdir = NULL,
  branch = NULL,
  incremental = FALSE
)
```

**Arguments**

root	A bundle directory path, a git URL, a tar/zip path or URL, or a bundle list from [okf_read()]. Non-directory sources are fetched via [okf_fetch()] and cleaned up afterwards.
db_path	DuckDB path; defaults to in-memory "":memory:"".
ingested_at	Optional ISO-8601 timestamp; defaults to the current time.
bundle_id	Optional stable bundle id.
source_kind	How the bundle was obtained (e.g. "dir"); auto-set for fetched sources.
subdir	Optional bundle path within a cloned/extracted source.
branch	Optional git branch or tag (git sources only).
incremental	Only rewrite concepts whose 'content_hash' changed since a prior ingest of the same bundle into 'db_path' (added/removed handled too); links and validation are always recomputed (they are graph-global). Falls back to a full load if the bundle is not already present. The 'summary' then includes 'changed'/'added'/'removed'/'cached' counts.

**Value**

A list with the open 'con', the 'bundle\_id', and a 'summary' (counts, conformance, link totals). The caller owns/closes 'con'.

---

okf_links	<i>Build the concept graph (resolved and broken links) for a bundle.</i>
-----------	--

---

**Description**

Build the concept graph (resolved and broken links) for a bundle.

**Usage**

```
okf_links(rd)
```

**Arguments**

rd                    A bundle as returned by [okf\_read()].

**Value**

A data.frame with 'src\_path', 'dst\_raw', 'dst\_path', 'resolved'.

---

okf_ollama_embedder	<i>Build an embedder backed by a local Ollama embeddings model.</i>
---------------------	---

---

**Description**

An embedder is a function of 'texts' returning a list of numeric vectors. Swap in any such function (e.g. an OpenAI client) for [okf\_embed()] / [okf\_rag()].

**Usage**

```
okf_ollama_embedder(
  model = "nomic-embed-text",
  url = Sys.getenv("OLLAMA_URL", "http://localhost:11434")
)
```

**Arguments**

model                Ollama embedding model name.  
url                    Ollama base URL (defaults to the 'OLLAMA\_URL' env var or localhost).

**Value**

A function 'texts -> list(numeric)'. Requires the httr2 package.

---

okf_parse_file	<i>Parse the YAML frontmatter and body of a single OKF concept file.</i>
----------------	--

---

**Description**

Parse the YAML frontmatter and body of a single OKF concept file.

**Usage**

```
okf_parse_file(path)
```

**Arguments**

path	Path to a markdown file.
------	--------------------------

**Value**

A list with 'meta' (parsed frontmatter, or 'NULL'), 'body', and 'err' ('NA' on success, else "no\_frontmatter", "unclosed\_frontmatter", or "yaml\_parse\_error").

---

okf_query	<i>Query helpers over an ingested OKF catalog.</i>
-----------	--

---

**Description**

Query helpers over an ingested OKF catalog.

**Usage**

```
okf_concepts(con)
```

```
okf_graph_df(con)
```

```
okf_findings(con)
```

```
okf_search(con, term)
```

**Arguments**

con	An open DuckDB connection to an okf catalog.
term	Search term for [okf_search()] (matched against concept bodies).

**Value**

A data.frame: concepts ([okf\_concepts]), link edges ([okf\_graph\_df]), validation findings ([okf\_findings]), or body matches ([okf\_search]).

---

okf_rag	<i>Semantic search over an embedded catalog.</i>
---------	--

---

**Description**

Embeds ‘query’ and returns the top-k most cosine-similar chunks (via DuckDB’s native ‘list\_cosine\_similarity’).  
Run [okf\_embed()] first.

**Usage**

```
okf_rag(con, query, embedder = NULL, k = 5L)
```

**Arguments**

con	An open DuckDB connection to an embedded okf catalog.
query	Query string.
embedder	An embedder function; defaults to [okf_ollama_embedder()].
k	Number of results to return.

**Value**

A data.frame with ‘path’, ‘title’, ‘chunk\_id’, ‘score’, ‘text’.

---

okf_read	<i>Read an OKF bundle from a directory into an in-memory representation.</i>
----------	--

---

**Description**

Read an OKF bundle from a directory into an in-memory representation.

**Usage**

```
okf_read(root, bundle_id = NULL, source_kind = "dir")
```

**Arguments**

root	Path to the bundle directory.
bundle_id	Optional stable id; defaults to a hash of the root path.
source_kind	How the bundle was obtained (e.g. “dir”).

**Value**

A list with ‘bundle\_id’, ‘root’, ‘okf\_version’, ‘source\_kind’, ‘concepts’ (parsed per-file records), and ‘known’ (all concept paths).

---

okf_resolve_link	<i>Resolve a markdown link target to a bundle-relative concept path.</i>
------------------	--

---

**Description**

Resolve a markdown link target to a bundle-relative concept path.

**Usage**

```
okf_resolve_link(raw, src_rel, known)
```

**Arguments**

raw	Raw link target.
src_rel	Bundle-relative path of the linking concept.
known	Character vector of all known concept paths in the bundle.

**Value**

The resolved bundle-relative path, or 'NA' if it does not resolve.

---

okf_validate	<i>Validate a bundle against the OKF v0.1 conformance rules (permissively).</i>
--------------	---

---

**Description**

Hard rules (severity 'error'): parseable frontmatter, non-empty 'type'. Soft findings (severity 'warn'): missing recommended fields, non-ISO timestamps, broken links. Never rejects the bundle – returns findings.

**Usage**

```
okf_validate(rd)
```

**Arguments**

rd	A bundle as returned by [okf_read()].
----	---------------------------------------

**Value**

A data.frame with 'path', 'severity', 'rule', 'message'.

# Index

[okf\\_backlinks](#), 2  
[okf\\_chunk\\_body](#), 3  
[okf\\_clusters](#), 3  
[okf\\_concepts \(okf\\_query\)](#), 12  
[okf\\_context](#), 4  
[okf\\_doctor](#), 4  
[okf\\_doctor\\_fix](#), 5  
[okf\\_embed](#), 6  
[okf\\_extract\\_links](#), 6  
[okf\\_fetch](#), 7  
[okf\\_findings \(okf\\_query\)](#), 12  
[okf\\_graph\\_df \(okf\\_query\)](#), 12  
[okf\\_graph\\_html](#), 7  
[okf\\_graph\\_json](#), 8  
[okf\\_graph\\_mermaid](#), 8  
[okf\\_html](#), 9  
[okf\\_impact](#), 9  
[okf\\_ingest](#), 10  
[okf\\_links](#), 11  
[okf\\_ollama\\_embedder](#), 11  
[okf\\_parse\\_file](#), 12  
[okf\\_query](#), 12  
[okf\\_rag](#), 13  
[okf\\_read](#), 13  
[okf\\_resolve\\_link](#), 14  
[okf\\_search \(okf\\_query\)](#), 12  
[okf\\_validate](#), 14