Package: quak (via r-universe)

June 10, 2026

Title Query 'Azure Data Lake Storage Gen2' with 'DuckDB'

Version 0.1.0

Description Provides convenience utilities for using 'DuckDB' directly over datasets stored in 'Azure Data Lake Storage Gen2' (ADLS Gen2, 'abfss://'). Opens connections configured for Azure-backed 'Delta Lake' and 'Parquet' data, registers Azure credentials as 'DuckDB' secrets, and supports optional repository mirrors for restricted networks. Integrates well with 'DBI' for SQL workflows and with 'dplyr' and 'dbplyr' for lazy table queries.

URL https://github.com/pedrobtz/quak

BugReports https://github.com/pedrobtz/quak/issues

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports cli, curl, DBI, duckdb, fs, glue, rlang, tools, utils

Suggests azr (>= 0.3.4), dbplyr, dplyr, testthat (>= 3.0.0), tibble, withr

Config/testthat/edition 3

Collate 'options.R' 'conditions.R' 'connection.R' 'cache.R' 'repositories.R' 'extensions.R' 'azure.R' 'datasets.R' 'tables.R' 'lake.R' 'delta.R' 'quak-package.R' 'zzz.R'

Config/pak/sysreqs cmake make libuv1-dev libssl-dev xz-utils

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

Date/Publication 2026-06-04 20:55:32 UTC

RemoteUrl https://github.com/pedrobtz/quak

RemoteRef HEAD

RemoteSha 6a39eb893aad9059537a4a69da47dbca80ba815a

Contents

az_conn — *Open a DuckDB connection configured for Azure Data Lake Storage Gen2*

Description

Opens a DuckDB connection and installs the `azure` and `delta` extensions. No secret is registered — use `az_set_token_secret()`, `az_set_sp_secret()`, or `az_set_chain_secret()` to supply credentials afterwards.

Usage

```
az_conn(conn = NULL)
```

Arguments

conn	An existing DuckDB connection to configure. When `NULL` (default) a new in-memory connection is opened via `conn_open()`.

Value

A DuckDB connection. The caller owns its lifetime; disconnect with `DBI::dbDisconnect(conn, shutdown = TRUE)`.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn() |>
  az_set_token_secret(token = my_token)
DBI::dbDisconnect(conn, shutdown = TRUE)

## End(Not run)
```

az_conn_settings — *Get Azure settings from a DuckDB connection*

Description

Queries `duckdb_settings()` and returns all entries whose name contains `"azure"`.

Usage

```
az_conn_settings(conn = az_conn())
```

Arguments

conn	A DuckDB connection. Defaults to az_conn().

Value

A tibble::tibble() with columns name, value, description.

Examples

```
conn <- DBI::dbConnect(duckdb::duckdb())
az_conn_settings(conn)
DBI::dbDisconnect(conn, shutdown = TRUE)
```

---

az_copy_to *Copy data to Azure Data Lake Storage Gen2*

Description

Writes a lazy table, data frame, or SQL query to an abfs:// or abfss:// URL using DuckDB's COPY ... TO command.

Usage

```
az_copy_to(
  conn,
  x,
  url,
  format = c("parquet", "csv", "json"),
  partition_by = NULL,
  overwrite = FALSE
)
```

Arguments

conn	A DuckDB connection.
x	A lazy dbplyr table, data frame, SQL string, or DBI::SQL object.
url	Character scalar. Azure Blob URL to write to.
format	Output format. One of "parquet", "csv", or "json".
partition_by	Optional character vector of columns to partition by.
overwrite	Logical. When TRUE, passes DuckDB's OVERWRITE_OR_IGNORE copy option.

Value

Invisibly returns url.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
az_copy_to(
  conn,
  "SELECT * FROM events WHERE event_date >= DATE '2026-01-01'",
  "abfss://container@account/exports/events",
  format = "parquet"
)

## End(Not run)
```

`az_default_scope` *Get the default Azure OAuth scope*

Description

Returns the Azure OAuth scope used in examples and token-based authentication helpers. Configure it with `options(quak.default_scope = "...")` or the `QUAK_DEFAULT_SCOPE` environment variable.

Usage

```
az_default_scope()
```

Value

A character scalar OAuth scope.

Examples

```
az_default_scope()
```

`az_delta_files` *List files in a Delta table on Azure Data Lake Storage Gen2*

Description

Returns DuckDB's `delta_list_files()` output for a Delta table.

Usage

```
az_delta_files(conn, url)
```

Arguments

conn	A DuckDB connection.
url	Character scalar. Azure Blob URL pointing to a Delta table.

Value

A tibble-like data frame with the active file manifest.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
az_delta_files(conn, "abfss://container@account/tables/sales")

## End(Not run)
```

---

az_exists *Check whether data exists at an Azure path*

Description

For an exact file or glob pattern, checks whether DuckDB's `glob()` returns at least one match. For a plain path, also probes `url/**` so dataset prefixes count as existing when they contain at least one object.

Usage

```
az_exists(conn, url)
```

Arguments

conn	A DuckDB connection.
url	Character scalar. Azure Blob URL or glob pattern.

Value

Logical scalar.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
az_exists(conn, "abfss://container@account/data/sales")

## End(Not run)
```

az_glimpse *Preview an Azure dataset*

Description

Prints a small preview and invisibly returns it as a tibble-like data frame.

Usage

```
az_glimpse(conn, url, n = 10, format = NULL)
```

Arguments

conn	A DuckDB connection.
url	Character scalar. Azure Blob URL.
n	Number of rows to preview. Default 10.
format	Optional format override. One of "parquet", "csv", "json", or "delta". When NULL, inferred from url.

Value

Invisibly returns the preview tibble-like data frame.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
az_glimpse(conn, "abfss://container@account/data/*.parquet", n = 5)

## End(Not run)
```

az_glob *List Azure paths matching a glob pattern*

Description

Uses DuckDB's glob() table function over Azure storage.

Usage

```
az_glob(conn, pattern)
```

Arguments

conn	A DuckDB connection.
pattern	Character scalar. abfs:// or abfss:// glob pattern.

Value

Character vector of matching paths.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
az_glob(conn, "abfss://container@account/data/*.parquet")

## End(Not run)
```

---

az_list_secrets *List Azure secrets registered in DuckDB*

Description

Queries duckdb_secrets() and returns secrets whose type is "azure". Values are returned as DuckDB reports them; DuckDB handles redaction of sensitive fields.

Usage

```
az_list_secrets(conn = conn_default())
```

Arguments

conn	A DuckDB connection. Defaults to conn_default().

Value

A tibble::tibble() with the columns returned by duckdb_secrets().

Examples

```
conn <- DBI::dbConnect(duckdb::duckdb())
az_list_secrets(conn)
DBI::dbDisconnect(conn, shutdown = TRUE)
```

az_schema *Inspect a dataset schema without collecting data*

Description

Uses DuckDB's `DESCRIBE SELECT` over a remote scan and returns only column names and DuckDB types.

Usage

```
az_schema(conn, url, format = NULL)
```

Arguments

`conn`	A DuckDB connection.
`url`	Character scalar. Azure Blob URL.
`format`	Optional format override. One of `"parquet"`, `"csv"`, `"json"`, or `"delta"`. When `NULL`, inferred from `url`.

Value

A tibble-like data frame with columns `name` and `type`.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
az_schema(conn, "abfss://container@account/data/*.parquet")

## End(Not run)
```

az_set_chain_secret *Register an Azure credential-chain secret*

Description

Creates or replaces a DuckDB Azure secret using the `credential_chain` provider. This lets DuckDB resolve credentials itself, for example from the Azure CLI or environment.

Usage

```
az_set_chain_secret(conn, account = NULL, chain = "default")
```

Arguments

conn	A DuckDB connection.
account	Optional storage account name. When supplied, the secret is scoped to that account.
chain	Optional character vector of DuckDB credential-chain entries. Values are joined with semicolons and passed as DuckDB's CHAIN value. Defaults to "default", DuckDB's default credential chain.

Value

Invisibly returns conn.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
az_set_chain_secret(conn, chain = "cli")

## End(Not run)
```

---

az_set_sp_secret *Register an Azure service-principal secret*

Description

Creates or replaces a DuckDB Azure secret using the service_principal provider.

Usage

```
az_set_sp_secret(conn, tenant_id, client_id, client_secret, account = NULL)
```

Arguments

conn	A DuckDB connection.
tenant_id	Character scalar. Azure Entra tenant ID.
client_id	Character scalar. Service principal client ID.
client_secret	Character scalar. Service principal client secret.
account	Optional storage account name. When supplied, the secret is scoped to that account.

Value

Invisibly returns conn.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
az_set_sp_secret(
  conn,
  tenant_id = "00000000-0000-0000-0000-000000000000",
  client_id = Sys.getenv("AZURE_CLIENT_ID"),
  client_secret = Sys.getenv("AZURE_CLIENT_SECRET")
)

## End(Not run)
```

---

az_set_token_secret *Register an Azure token secret*

Description

Creates or replaces a DuckDB Azure secret using the `access_token` provider. Use this when another package has already obtained an access token and you want to register or refresh a token secret.

Usage

```
az_set_token_secret(conn, token, account = NULL)
```

Arguments

`conn`	A DuckDB connection.
`token`	Character scalar. Access token value.
`account`	Optional storage account name. When supplied, the secret is scoped to `abfss://<account>/`.

Value

Invisibly returns `conn`.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
az_set_token_secret(conn, token = "<access-token>")

## End(Not run)
```

az_tune *Tune Azure read settings on a DuckDB connection*

Description

Sets the Azure performance and transport settings exposed by DuckDB. Each argument defaults to NULL, which leaves that setting unchanged.

Usage

```
az_tune(
  conn,
  concurrency = NULL,
  chunk_size = NULL,
  buffer_size = NULL,
  transport = NULL,
  metadata_cache = NULL,
  context_cache = NULL
)
```

Arguments

conn	A DuckDB connection.
concurrency	Optional positive whole number for azure_read_transfer_concurrency.
chunk_size	Optional positive whole number or character scalar for azure_read_transfer_chunk_size.
buffer_size	Optional positive whole number or character scalar for azure_read_buffer_size.
transport	Optional character scalar for azure_transport_option_type.
metadata_cache	Optional logical scalar for enable_http_metadata_cache.
context_cache	Optional logical scalar for azure_context_caching.

Value

Invisibly returns conn.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
az_tune(conn, concurrency = 8, metadata_cache = TRUE)

## End(Not run)
```

az_write_parquet *Write Parquet data to Azure Data Lake Storage Gen2*

Description

Thin convenience wrapper around `az_copy_to()` with `format = "parquet"`.

Usage

```
az_write_parquet(conn, x, url, partition_by = NULL, overwrite = FALSE)
```

Arguments

`conn`	A DuckDB connection.
`x`	A lazy dbplyr table, data frame, SQL string, or `DBI::SQL` object.
`url`	Character scalar. Azure Blob URL to write to.
`partition_by`	Optional character vector of columns to partition by.
`overwrite`	Logical. When `TRUE`, passes DuckDB's `OVERWRITE_OR_IGNORE` copy option.

Value

Invisibly returns `url`.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
az_write_parquet(conn, data.frame(x = 1:3), "abfss://container@account/x")

## End(Not run)
```

collect.tbl_az *Collect an Azure-backed lazy tbl*

Description

`dplyr::collect()` method for tables created by `tbl_delta()` and `tbl_parquet()`. Verifies that the backing DuckDB connection is still open and that the `azure` extension is loaded before the query is materialised, then defers to the underlying dbplyr method.

Usage

```
## S3 method for class 'tbl_az'
collect(x, ...)
```

Arguments

x	A tbl_az produced by tbl_delta() or tbl_parquet().
...	Passed on to the next collect() method.

Value

A tibble::tibble() with the collected rows.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
tbl_delta(conn, "abfss://container@account/path/sales") |>
  dplyr::collect()

## End(Not run)
```

---

conn_setting *Get or set DuckDB settings*

Description

When called with no arguments, returns all settings as a data frame. When name is supplied and value is NULL, returns the value of that setting. When both name and value are supplied, executes SET <name> = <value>.

Usage

```
conn_setting(conn = conn_default(), name = NULL, value = NULL)
```

Arguments

conn	A DuckDB connection.
name	Optional character scalar. Setting name.
value	Optional value to set. Coerced to character; DuckDB casts it to the appropriate type.

Value

All settings: a tibble::tibble(). Single setting read: a character scalar. Write: conn invisibly.

Examples

```
conn <- DBI::dbConnect(duckdb::duckdb())
conn_setting(conn, "threads")
DBI::dbDisconnect(conn, shutdown = TRUE)
```

ext_cache *Extension cache*

Description

Builds an ext_cache object: a list of closures bound to a cache directory, implementing CRUD over cached .duckdb_extension files. Files are laid out under <cache_path>/<version>/<platform>/<name>.duckdb_exten

Usage

```
ext_cache(cache_path = ext_cache_path())
```

Arguments

cache_path	Character scalar. Cache root directory. Defaults to ext_cache_path().

Value

An ext_cache object (a list of closures) with elements:

- .path: the cache root.
- get(name, version, platform): path to the cached extension, or NULL.
- add(name, version, platform, src): copies src into the cache.
- list(): data frame of cached extensions.
- del(name, version, platform): removes a cached extension. When version and platform are omitted, removes all cached entries for name.

Examples

```
cache <- ext_cache(file.path(tempdir(), "quak-cache"))
cache$.path
```

ext_cache_path *Default DuckDB extension cache directory*

Description

Resolution order: in-memory value (opts$set("cache_dir", ...)) -> env var QUAK_CACHE_DIR -> OS-appropriate user cache directory via tools::R_user_dir().

Usage

```
ext_cache_path()
```

Value

Character scalar. The resolved cache path.

Examples

```
ext_cache_path()
```

`ext_dir` *Find the DuckDB extension folder*

Description

Returns the path where DuckDB stores installed extension files. This is determined by the `extension_directory` setting.

Usage

```
ext_dir(conn = conn_default())
```

Arguments

`conn`	A DuckDB connection. Defaults to `conn_default()`.

Value

Character scalar. Path to the extension directory.

Examples

```
conn <- DBI::dbConnect(duckdb::duckdb())
ext_dir(conn)
DBI::dbDisconnect(conn, shutdown = TRUE)
```

`ext_install` *Install a DuckDB extension*

Description

Tries two strategies in order, succeeding as soon as one works:

Usage

```
ext_install(
  name,
  cache = ext_cache(),
  repo = c("core", "community"),
  conn = conn_default(),
  verbose = NULL
)
```

Arguments

name	Character scalar. Extension name.
cache	An ext_cache object used by the manual fallback.
repo	"core" or "community". Determines which configured URL to use and, when no URL is set, which DuckDB install syntax to emit.
conn	A DuckDB connection. Defaults to conn_default().
verbose	Logical or NULL. When TRUE, emits a warning if the SQL install fails but the manual fallback succeeds. When NULL (default), uses the quak.install_verbose option / QUAK_INSTALL_VERBOSE env var. When FALSE, the fallback is silent. Either way, a SQL failure is never raised as an error on its own.

Details

1. **SQL install:** runs DuckDB's built-in INSTALL (using the configured repository URL when one is set via repo_set_urls(), the QUAK_CORE_REPO / QUAK_COMMUNITY_REPO env vars, or the quak.core_repo / quak.community_repo R options).
2. **Manual fallback:** when the SQL install fails (e.g. DuckDB cannot reach an HTTPS URL before httpfs is loaded, whereas R's curl can), downloads the .duckdb_extension file, caches it, and copies it into the extension directory.

A SQL failure is never raised on its own — it only surfaces (as a warning, when verbose = TRUE) if the manual fallback also runs. An error is raised only when both strategies fail.

Idempotent — skips install if the extension is already installed (checked via the duckdb_extensions() pragma).

Value

Invisibly returns conn.

Examples

```
## Not run:
# Requires network access to download the extension.
conn <- DBI::dbConnect(duckdb::duckdb())
ext_install("httpfs", conn = conn)
DBI::dbDisconnect(conn, shutdown = TRUE)

## End(Not run)
```

---

ext_install_local *Install a DuckDB extension from a local file*

Description

Executes INSTALL '/path/to/ext.duckdb_extension' on conn. Use this to install an extension binary you already have on disk without going through a remote repository.

Usage

```
ext_install_local(path, name = NULL, conn = conn_default())
```

Arguments

path	Character scalar. Path to the .duckdb_extension file.
name	Character scalar. Extension name used in messages. Inferred from path when omitted.
conn	A DuckDB connection. Defaults to conn_default().

Value

Invisibly returns conn.

Examples

```
## Not run:
# Requires a local DuckDB extension file at the given path.
conn <- DBI::dbConnect(duckdb::duckdb())
ext_install_local("/path/to/httpfs.duckdb_extension", conn = conn)
DBI::dbDisconnect(conn, shutdown = TRUE)

## End(Not run)
```

---

ext_is_installed *Check whether a DuckDB extension is installed*

Description

Check whether a DuckDB extension is installed

Usage

```
ext_is_installed(name, conn = conn_default())
```

Arguments

name	Character scalar. Extension name.
conn	A DuckDB connection. Defaults to conn_default().

Value

Logical scalar.

Examples

```
conn <- DBI::dbConnect(duckdb::duckdb())
ext_is_installed("httpfs", conn = conn)
DBI::dbDisconnect(conn, shutdown = TRUE)
```

ext_list_available *List all DuckDB core extensions*

Description

Returns the full catalog of extensions maintained by the DuckDB core team, regardless of whether they are installed.

Usage

```
ext_list_available(conn = conn_default())
```

Arguments

conn	A DuckDB connection. Defaults to `conn_default()`.

Value

A `tibble::tibble()` with columns: `name`, `version`, `description`.

Examples

```
conn <- DBI::dbConnect(duckdb::duckdb())
ext_list_available(conn)
DBI::dbDisconnect(conn, shutdown = TRUE)
```

ext_list_installed *List installed DuckDB extensions*

Description

Queries `duckdb_extensions()`, returning only extensions where `installed = TRUE`.

Usage

```
ext_list_installed(conn = conn_default())
```

Arguments

conn	A DuckDB connection. Defaults to `conn_default()`.

Value

A `tibble::tibble()` with columns: `name`, `installed`, `loaded`, `version`, `description`.

Examples

```
conn <- DBI::dbConnect(duckdb::duckdb())
ext_list_installed(conn)
DBI::dbDisconnect(conn, shutdown = TRUE)
```

`ext_load` *Load a DuckDB extension, installing it first if necessary*

Description

When `path` is supplied, executes `LOAD '/path/to/ext.duckdb_extension'` directly — no install check or auto-install occurs. When only `name` is supplied, returns immediately if the extension is already loaded. Otherwise it checks whether the extension is installed; if not and `auto_install = TRUE`, installs it (prompting first when `ask = TRUE` and the session is interactive), then executes `LOAD <name>`.

Usage

```
ext_load(
  name = NULL,
  path = NULL,
  conn = conn_default(),
  auto_install = TRUE,
  ask = rlang::is_interactive(),
  cache = ext_cache(),
  repo = c("core", "community")
)
```

Arguments

`name`	Character scalar. Extension name. When `path` is supplied, `name` is inferred from the filename and used only in messages.
`path`	Optional character scalar. Path to a local `.duckdb_extension` file. When supplied, the extension is loaded directly from disk, bypassing the install check and `ext_install()`.
`conn`	A DuckDB connection. Defaults to `conn_default()`.
`auto_install`	Logical. Install automatically when the extension is missing. Default `TRUE`. Ignored when `path` is supplied.
`ask`	Logical. Prompt the user before installing. Defaults to `rlang::is_interactive()`, so it never prompts during tests or in non-interactive sessions. Ignored when `auto_install = FALSE` or `path` is supplied.
`cache`	An `ext_cache` object forwarded to `ext_install()` on auto-install. Ignored when `path` is supplied.
`repo`	`"core"` or `"community"`. Forwarded to `ext_install()`. Ignored when `path` is supplied.

Value

Invisibly returns conn.

Examples

```
## Not run:
# Requires network access to download and load the extension.
conn <- DBI::dbConnect(duckdb::duckdb())
ext_load("httpfs", conn = conn)
DBI::dbDisconnect(conn, shutdown = TRUE)

## End(Not run)
```

---

ext_set_dir *Set the DuckDB extension folder*

Description

Changes the path where DuckDB stores installed extension files for conn. The value is written to DuckDB's extension_directory setting.

Usage

```
ext_set_dir(path, conn = conn_default(), create = TRUE)
```

Arguments

path	Character scalar. Path to the extension directory.
conn	A DuckDB connection. Defaults to conn_default().
create	Logical. If TRUE, create path before setting it.

Value

Invisibly returns the normalized extension directory path.

Examples

```
conn <- DBI::dbConnect(duckdb::duckdb())
ext_set_dir(file.path(tempdir(), "quak-exts"), conn = conn)
DBI::dbDisconnect(conn, shutdown = TRUE)
```

`ext_uninstall` *Uninstall a DuckDB extension*

Description

Removes the extension file from DuckDB's `extension_directory`. Optionally also purges the corresponding entry from the local cache.

Usage

```
ext_uninstall(
  name,
  purge_cache = FALSE,
  cache = ext_cache(),
  conn = conn_default()
)
```

Arguments

`name`	Character scalar. Extension name.
`purge_cache`	Logical. If `TRUE`, also removes the file from `cache`.
`cache`	An `ext_cache` object. Only used when `purge_cache = TRUE`.
`conn`	A DuckDB connection. Defaults to `conn_default()`.

Value

Invisibly returns `conn`.

Examples

```
## Not run:
# Requires a connection with the extension already installed.
conn <- DBI::dbConnect(duckdb::duckdb())
ext_uninstall("httpfs", conn = conn)
DBI::dbDisconnect(conn, shutdown = TRUE)

## End(Not run)
```

load_csv — *Register a CSV dataset as a view on a DuckDB connection*

Description

Validates the URL, loads the azure extension, then registers the dataset as a VIEW over read_csv_auto(). Use az_conn() first if the connection needs an Azure secret. Returns conn invisibly — use tbl_csv() if you want a dplyr::tbl().

Usage

```
load_csv(conn, url, name, replace = TRUE, ...)
```

Arguments

conn	A DuckDB connection.
url	Character scalar. Azure Blob URL. Supports glob patterns.
name	Character scalar. Name to register the view under in DuckDB.
replace	Logical. Replace an existing view. Default TRUE.
...	Reader options forwarded to DuckDB's read_csv_auto().

Value

Invisibly returns conn.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
load_csv(conn, "abfss://container@account/data/*.csv", name = "events")

## End(Not run)
```

load_dataset — *Register a Delta, Parquet, CSV, or JSON dataset on a DuckDB connection*

Description

Dispatches to load_delta(), load_parquet(), load_csv(), or load_json() based on format. Only arguments accepted by the target function may be passed via ...; passing format-incompatible arguments raises an error.

Usage

```
load_dataset(
  conn,
  url,
  name,
  format = c("delta", "parquet", "csv", "json"),
  ...
)
```

Arguments

conn	A DuckDB connection.
url	Character scalar. Azure Blob URL.
name	Character scalar. Name to register the dataset under in DuckDB.
format	One of "delta", "parquet", "csv", or "json".
...	Passed to the selected loader.

Value

Invisibly returns conn.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
load_dataset(
  conn,
  "abfss://container@account/path/sales",
  name = "sales",
  format = "delta"
)

## End(Not run)
```

---

load_delta *Register a Delta Lake table on a DuckDB connection*

Description

Validates the URL, loads the azure and delta extensions, then registers the table either as an ATTACH database or a VIEW. Use az_conn() first if the connection needs an Azure secret. Returns conn invisibly — use tbl_delta() if you want a dplyr::tbl().

Usage

```
load_delta(
  conn,
  url,
  name,
  method = c("attach", "view"),
  replace = TRUE,
  version = NULL,
  timestamp = NULL
)
```

Arguments

conn	A DuckDB connection.
url	Character scalar. Azure Blob URL pointing to a Delta table.
name	Character scalar. Name to register the table under in DuckDB.
method	"attach" (default) or "view".
replace	Logical. Replace an existing registration. Default TRUE.
version	Optional non-negative Delta table version to attach.
timestamp	Optional Delta table timestamp to attach. Only one of version and timestamp may be supplied.

Value

Invisibly returns conn.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
load_delta(conn, "abfss://container@account/path/sales", name = "sales")
DBI::dbGetQuery(conn, "SELECT COUNT(*) FROM sales")

## End(Not run)
```

---

load_json *Register a JSON dataset as a view on a DuckDB connection*

Description

Validates the URL, loads the azure extension, then registers the dataset as a VIEW over read_json_auto(). Use az_conn() first if the connection needs an Azure secret. Returns conn invisibly — use tbl_json() if you want a dplyr::tbl().

Usage

```
load_json(conn, url, name, replace = TRUE, ...)
```

Arguments

conn	A DuckDB connection.
url	Character scalar. Azure Blob URL. Supports glob patterns.
name	Character scalar. Name to register the view under in DuckDB.
replace	Logical. Replace an existing view. Default TRUE.
...	Reader options forwarded to DuckDB's read_json_auto().

Value

Invisibly returns conn.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
load_json(conn, "abfss://container@account/data/*.json", name = "events")

## End(Not run)
```

---

load_parquet *Register a Parquet dataset as a view on a DuckDB connection*

Description

Validates the URL, loads the azure extension, then registers the dataset as a VIEW. Use az_conn() first if the connection needs an Azure secret. Returns conn invisibly — use tbl_parquet() if you want a dplyr::tbl().

Usage

```
load_parquet(conn, url, name, hive_partitioning = FALSE, replace = TRUE)
```

Arguments

conn	A DuckDB connection.
url	Character scalar. Azure Blob URL. Supports glob patterns.
name	Character scalar. Name to register the view under in DuckDB.
hive_partitioning	Logical. Enable Hive partition inference. Default FALSE.
replace	Logical. Replace an existing view. Default TRUE.

Value

Invisibly returns conn.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
load_parquet(conn, "abfss://container@account/data/*.parquet", name = "events")

## End(Not run)
```

`print.quak_opts` *Print the quak option registry*

Description

Renders one row per option with its current (resolved) value, the source that value came from, the environment variable that can override it (and whether it is set), and the built-in default.

Usage

```
## S3 method for class 'quak_opts'
print(x, mask = TRUE, ...)
```

Arguments

`x`	A `quak_opts` object (the internal `opts` registry).
`mask`	Logical. When `TRUE` (default), sensitive option values are shown as `"<hidden>"` when set.
`...`	Unused.

Value

Invisibly returns `x`.

quak_options *List all quak options and their current values*

Description

Prints every quak option (via `print.quak_opts()`) and invisibly returns a tibble of the same information. The resolution order is: value set via `options(quak.*)` -> the option's env var -> a built-in default.

Usage

```
quak_options(mask = TRUE)
```

Arguments

`mask`	Logical. When `TRUE` (default), sensitive option values are shown as `"<hidden>"` when set.

Value

Invisibly, a `tibble::tibble()` with columns `option`, `value`, `source`, `env_var`, `env_value`, and `default`.

Examples

```
quak_options()
```

repo_set_urls *Set DuckDB extension repository URLs*

Description

Stores URLs in R options `quak.core_repo` / `quak.community_repo` so they can be configured org-wide in `.Rprofile`. When `core` is supplied, also sets DuckDB's `custom_extension_repository` on `conn`; passing `NULL` resets that connection setting to DuckDB's default.

Usage

```
repo_set_urls(
  core = NULL,
  community = NULL,
  check = TRUE,
  conn = conn_default()
)
```

Arguments

core	Optional character scalar. URL for the core extension repository. Omit to leave the current value unchanged. Pass NULL to reset to the DuckDB default.
community	Optional character scalar. URL for the community extension repository. Omit to leave the current value unchanged. Pass NULL to reset to the DuckDB default.
check	Logical. If TRUE (default), calls repo_check() for each repository whose URL was changed, probing "httpfs" as a baseline extension.
conn	A DuckDB connection. Defaults to conn_default(). Used to set custom_extension_repository when core is supplied, and by repo_check() when check = TRUE.

Value

Invisibly returns a named list with elements core and community reflecting the current option values.

Examples

```
old <- repo_urls()
repo_set_urls(core = "https://extensions.example.com", check = FALSE)
repo_urls()
repo_set_urls(core = old$core, check = FALSE)
```

---

repo_urls *Get DuckDB extension repository URLs*

Description

Returns the currently active repository URLs. Resolution order per repo: R option (quak.core_repo / quak.community_repo) -> env var (QUAK_CORE_REPO / QUAK_COMMUNITY_REPO) -> built-in default.

Usage

```
repo_urls()
```

Value

A named list with elements core and community.

Examples

```
repo_urls()
```

tbl_csv *Open a CSV dataset as a lazy dplyr tbl*

Description

Validates the URL, loads the azure extension, then returns a lazy dplyr::tbl() over the dataset. Use az_conn() first if the connection needs Azure extensions, settings, or secrets.

Usage

```
tbl_csv(conn, url, name = NULL, replace = TRUE, ...)
```

Arguments

conn	A DuckDB connection.
url	Character scalar. Azure Blob URL. Supports glob patterns.
name	Optional character scalar. Name to register the view under in DuckDB. When NULL (default) the dataset is scanned directly.
replace	Logical. Replace an existing view of the same name. Default TRUE. Ignored when name = NULL.
...	Reader options forwarded to DuckDB's read_csv_auto().

Details

When name is NULL the dataset is queried directly via read_csv_auto() with no persistent object registered on the connection. When name is supplied the dataset is first registered as a VIEW via load_csv(), then referenced by name.

Value

A dplyr::tbl() backed by the CSV dataset.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
tbl_csv(conn, "abfss://container@account/data/*.csv") |>
  dplyr::collect()

## End(Not run)
```

tbl_delta *Open a Delta Lake table as a lazy dplyr tbl*

Description

Validates the URL, loads the azure and delta extensions, then returns a lazy dplyr::tbl() over the table. Use az_conn() first if the connection needs Azure extensions, settings, or secrets.

Usage

```
tbl_delta(
  conn,
  url,
  name = NULL,
  method = c("attach", "view"),
  replace = TRUE,
  version = NULL,
  timestamp = NULL
)
```

Arguments

conn	A DuckDB connection.
url	Character scalar. Azure Blob URL pointing to a Delta table (e.g. "abfss://container@account.dfs.co
name	Optional character scalar. Name to register the table under in DuckDB. When NULL (default) the table is scanned directly.
method	"attach" (default) or "view". Ignored when name = NULL.
replace	Logical. Replace an existing registration of the same name. Default TRUE. Ignored when name = NULL.
version	Optional non-negative Delta table version to read.
timestamp	Optional Delta table timestamp to read. Only one of version and timestamp may be supplied.

Details

When name is NULL the table is queried directly via delta_scan() with no persistent object registered on the connection. When name is supplied the table is first registered via load_delta() (as an ATTACH database or a VIEW depending on method), then referenced by name.

Delta time travel currently requires name because DuckDB exposes version and timestamp through ATTACH, not delta_scan().

Value

A dplyr::tbl() backed by the Delta table.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
tbl_delta(conn, "abfss://container@account/path/sales") |>
  dplyr::filter(amount > 100) |>
  dplyr::collect()

## End(Not run)
```

`tbl_json` *Open a JSON dataset as a lazy dplyr tbl*

Description

Validates the URL, loads the `azure` extension, then returns a lazy `dplyr::tbl()` over the dataset. Use `az_conn()` first if the connection needs Azure extensions, settings, or secrets.

Usage

```
tbl_json(conn, url, name = NULL, replace = TRUE, ...)
```

Arguments

`conn`	A DuckDB connection.
`url`	Character scalar. Azure Blob URL. Supports glob patterns.
`name`	Optional character scalar. Name to register the view under in DuckDB. When `NULL` (default) the dataset is scanned directly.
`replace`	Logical. Replace an existing view of the same name. Default `TRUE`. Ignored when `name = NULL`.
`...`	Reader options forwarded to DuckDB's `read_json_auto()`.

Details

When `name` is `NULL` the dataset is queried directly via `read_json_auto()` with no persistent object registered on the connection. When `name` is supplied the dataset is first registered as a VIEW via `load_json()`, then referenced by name.

Value

A `dplyr::tbl()` backed by the JSON dataset.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
tbl_json(conn, "abfss://container@account/data/*.json") |>
  dplyr::collect()

## End(Not run)
```

`tbl_parquet` *Open a Parquet dataset as a lazy dplyr tbl*

Description

Validates the URL, loads the `azure` extension, then returns a lazy `dplyr::tbl()` over the dataset. Use `az_conn()` first if the connection needs Azure extensions, settings, or secrets.

Usage

```
tbl_parquet(conn, url, name = NULL, hive_partitioning = FALSE, replace = TRUE)
```

Arguments

`conn`	A DuckDB connection.
`url`	Character scalar. Azure Blob URL. Supports glob patterns for multi-file datasets (e.g. `"abfss://container@account.dfs.core.windows.net/data/*.parquet"`).
`name`	Optional character scalar. Name to register the view under in DuckDB. When `NULL` (default) the dataset is scanned directly.
`hive_partitioning`	Logical. Enable Hive partition inference from the directory structure. Default `FALSE`.
`replace`	Logical. Replace an existing view of the same name. Default `TRUE`. Ignored when `name = NULL`.

Details

When `name` is `NULL` the dataset is queried directly via `read_parquet()` with no persistent object registered on the connection. When `name` is supplied the dataset is first registered as a VIEW via `load_parquet()`, then referenced by name. Glob patterns (e.g. `"*.parquet"`) are supported in `url` for multi-file datasets.

Value

A `dplyr::tbl()` backed by the Parquet dataset.

Examples

```
## Not run:
# Requires a live Azure account, credentials, and network access.
conn <- az_conn()
tbl_parquet(conn, "abfss://container@account/data/*.parquet") |>
  dplyr::collect()

## End(Not run)
```

Index