Package: airtable2 (via r-universe)

June 5, 2026

Title Pure R Client for the 'Airtable' REST API

Version 0.0.0.9000

Description A modern R client for the 'Airtable' REST API. Provides low-level wrappers for all API endpoints, high-level functions for reading, writing, upserting, and syncing records, and a DBI-compliant interface for RStudio/Positron connection pane integration.

License MIT + file LICENSE

URL https://github.com/noamross/airtable2, https://noamross.github.io/airtable2/

BugReports https://github.com/noamross/airtable2/issues

Depends R (>= 4.1.0)

Imports cli, DBI, digest, glue, httr2, jsonlite, methods, rlang (>= 1.0.0), tibble, vctrs

Suggests httptest2, knitr, pillar, rmarkdown, rstudioapi, spelling, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

RoxygenNote 8.0.0

Language en-US

VignetteBuilder knitr

Config/pak/sysreqs libssl-dev

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

Date/Publication 2026-06-05 17:59:38 UTC

RemoteUrl https://github.com/noamross/airtable2

RemoteRef HEAD

RemoteSha b23512c66243ed061e21bdd2293f99b6a1856a3b

Contents

`air_api_usage` *Report Airtable API usage for a workspace*

Description

Returns this session's best-effort tally of API calls made against a workspace during the current calendar month (UTC). Airtable free/team plans cap each workspace at roughly 1000 calls per month and provide no way to query the remaining quota, so `airtable2` keeps its own counter on disk.

Usage

```
air_api_usage(workspace = NULL)
```

Arguments

`workspace`	Workspace ID (starts with `wsp`). Defaults to the configured default workspace (`getOption("airtable2.workspace_id")` or `Sys.getenv("AIRTABLE_WORKSPACE_ID")`).

Value

A list of class `air_api_usage` with elements `workspace_id`, `count`, `since`, and `last`. Has a `{cli}` print method.

Examples

```
## Not run:
air_api_usage("wspXXXXXXXXXXXXXX")

## End(Not run)
```

`air_attachment_preview_url` *Get a stable preview URL for an Airtable attachment*

Description

The Airtable REST API returns a **temporary** download URL in the `url` field of every attachment object. This URL is served from `airtableusercontent.com` and **expires after approximately 2 hours**. Do not save or share the raw API `url` as a permanent link — it will stop working shortly after it is issued.

Usage

```
air_attachment_preview_url(attachment)
```

Arguments

`attachment`	A single attachment metadata list (as returned by `air_read(..., attachments = "meta")`) **or** a list of such attachment lists. Each element should contain fields such as `id`, `url`, `filename`, `size`, and `type`.

Value

A character vector of `NA_character_`, one element per attachment. The function always returns `NA` because a stable viewer URL cannot be constructed from API metadata alone. See the @section above for the correct manual workflow.

Stable attachment viewer URLs

Airtable maintains a separate **attachment viewer URL** (served from `airtable.com`) that does **not** expire (unless the attachment or record is deleted). This stable URL requires the recipient to have access to the base or interface in which the attachment lives.

Crucially, the stable viewer URL cannot be derived from any field that the REST API returns (`id`, `url`, `filename`, `size`, `type`, `thumbnails`). It is only obtainable by navigating to the record in the Airtable web app, clicking the attachment to open its preview, and copying the URL from the browser address bar.

Because no stable URL can be constructed programmatically from attachment metadata, `air_attachment_preview_url()` always returns `NA_character_` and emits an informative message directing users to the correct workflow.

Examples

```
# Suppose you have attachment metadata from air_read():
att <- list(
  id       = "attABCDEF123456",
  url      = "https://v5.airtableusercontent.com/v3/some_signed_path",
  filename = "report.pdf",
```

```
  size     = 204800L,
  type     = "application/pdf"
)

# The temporary download URL (expires in ~2 hours):
att$url

# Attempting to get a stable preview URL — always NA with an explanation:
## Not run:
air_attachment_preview_url(att)

## End(Not run)

# To obtain a permanent, shareable link for a colleague:
# 1. Open the record in the Airtable web app.
# 2. Click the attachment to open its viewer.
# 3. Copy the URL from the browser address bar (it is from airtable.com).
# 4. Share that URL — it does not expire as long as the attachment exists
#    and the recipient has access to the base.
```

`air_browse` *Open an Airtable workspace, base, table, or view in the browser*

Description

Navigates to the Airtable web interface for the given resource. Automatically resolves IDs from connection objects, determines the resource type from the ID prefix, or resolves human-readable names to IDs.

Usage

```
air_browse(id = NULL, base_id = NULL, table_id = NULL, ...)
```

Arguments

`id`	Character, connection, or `NULL`. One of: • `NULL` (default): opens the session default base. • A workspace ID (starts with `wsp`) • A base ID (starts with `app`) • A table ID (starts with `tbl`) • A view ID (starts with `viw`) • A record ID (starts with `rec`) • A full Airtable URL • An Airtable connection object (uses its `base_id`) • A human-readable base or table name (see Name resolution above)
`base_id`	Character or `NULL`. The base ID to use when browsing a table, view, or record, or when resolving a table name. Falls back to the session default set via `air_set_base()`.
`table_id`	Character or `NULL`. The table ID to use when browsing a view or record ID.
`...`	Additional arguments passed to `utils::browseURL()`.

Details

If `id` is `NULL` (the default), `air_browse()` opens the session default base set via `air_set_base()` or the `airtable2.base_id` option.

Name resolution precedence:

- If `id` is a human name **and** a `base_id` is available (explicit arg or session default), `id` is treated as a **table name** and resolved via `at_get_schema()`. Case-sensitive first; falls back to case-insensitive with a warning if needed.
- If `id` is a human name and **no** base context is available, `id` is treated as a **base name** and resolved via `at_list_bases()`.
- If no match is found, `cli_abort()` is called with a helpful message listing how to view valid names.

Value

The URL that was opened (invisibly).

Examples

```
## Not run:
# Open the session default base (requires air_set_base() or option)
air_browse()

# Open a base by ID
air_browse("appXXXXXX")

# Open a workspace
air_browse("wspXXXXXX")

# Open a table within a base (base_id is a proper formal, not ...)
air_browse("tblXXXXXX", base_id = "appXXXXXX")

# Open a view
air_browse("viwXXXXXX", base_id = "appXXXXXX", table_id = "tblXXXXXX")

# Resolve a table by name (requires base context)
air_browse("My Table", base_id = "appXXXXXX")

# Resolve a base by name (no base context)
air_browse("My Base")

## End(Not run)
```

`air_connect` *Connect to Airtable via DBI*

Description

A convenience wrapper around DBI::dbConnect() using the airtable2() driver. When base is specified, the connection shows that base's tables directly. When base is omitted, all accessible bases are shown as schemas in the connection pane. Use bases to restrict the pane to a specific subset of bases.

Usage

```
air_connect(
  base = NULL,
  bases = NULL,
  .token = NULL,
  include_views = FALSE,
  .connect_code = NULL
)
```

Arguments

base	Character. A Base ID (starts with app) or a Base Name. If NULL, connects to all accessible bases.
bases	Character vector of Base IDs or names. When supplied, the connection pane shows only those bases (as schemas). Cannot be combined with base.
.token	Character. Airtable Personal Access Token (resolved via air_token() if NULL).
include_views	Logical. If TRUE, views are included in the connection pane alongside tables. Default FALSE.
.connect_code	Character. Optional custom reconnect code for the IDE connection pane. Defaults to a DBI::dbConnect() call. Set by air_pane() to use airtable2::air_pane() instead.

Value

An AirtableConnection object.

Examples

```
## Not run:
# Connect to a specific base by name
con <- air_connect(base = "Project Tracker")

# Connect to all accessible bases
con <- air_connect()

# Show only selected bases in the pane
con <- air_connect(bases = c("appXXXXXX", "appYYYYYY"))

## End(Not run)
```

air_delete *Delete records from a table (high-level)*

Description

A convenience wrapper around `at_delete_records()` with messaging.

Usage

```
air_delete(record_ids, table, base_id = NULL, .token = NULL, progress = NULL)
```

Arguments

`record_ids`	Character vector of record IDs to delete.
`table`	Table name or ID.
`base_id`	Base ID (e.g., `"appXXXXXX"`). If `NULL`, uses the session default set by `air_set_base()` or the `AIRTABLE_BASE_ID` environment variable.
`.token`	Personal access token (resolved via `air_token()` if `NULL`).
`progress`	Logical or `NULL`. If `TRUE`, shows a cli progress bar for paginated requests. If `NULL` (default), uses option `airtable2.progress.bar` or env var `AIRTABLE2_PROGRESS_BAR`.

Value

Invisible `NULL`. Side effect: deletes records.

Examples

```
## Not run:
air_delete(c("recABC", "recDEF"), "Contacts", "appXXXXXX")

## End(Not run)
```

air_demo *Run an interactive airtable2 demo walkthrough*

Description

Runs through canonical airtable2 operations against a BollardsForArt demo base, pausing at each step so you can watch changes appear in the browser. If no `base_id` is provided and none is set as the session default, calls `air_demo_setup()` first.

Usage

```
air_demo(
  base_id = NULL,
  workspace_id = Sys.getenv("AIRTABLE_WORKSPACE_ID"),
  .token = NULL
)
```

Arguments

base_id	Base ID to use. If NULL, checks the session default (air_set_base()) and AIRTABLE_BASE_ID env var; calls air_demo_setup() if nothing is found.
workspace_id	Passed to air_demo_setup() if base creation is needed.
.token	API token (see air_set_token()).

Details

Open the Airtable base in a browser alongside the console. The walkthrough pauses with <Enter> prompts whenever you need to switch to a different table, and sleeps two seconds between steps so changes can appear.

Steps covered:

1. Read all Artists with a progress bar
2. Write one new artist; read back showing airtable_created_time
3. Write 120 community supporters in 12 batches – progress bar is clearly visible; read back over 2 pages to demonstrate read pagination
4. Bulk-upsert 30 artists with a progress bar
5. Sync back to the original 15 with a progress bar (watch deletions)
6. Upsert a new Engagement Score column into Artists from R
7. Upload an image attachment to a Project record
8. Link artists to projects via multipleRecordLinks
9. Left-join Airtable columns into a local R tibble
10. View the base schema
11. Seed a _metadata table with air_meta_init(), edit table/column names as rows in Airtable, then apply with air_meta_sync()
12. Connect via the DBI interface: DBI::dbConnect(), DBI::dbListTables(), DBI::dbReadTable(), DBI::dbWriteTable()
13. View API usage

All operations keyed on Name are idempotent; re-running will not accumulate duplicate records.

Value

Invisibly returns a list of results from each step.

Examples

```
## Not run:
# Run with an existing demo base (set via air_set_base() or env var)
air_set_base("appXXXXXXXXXXXXXX")
air_demo()

# Or create a new demo base first (needs AIRTABLE_WORKSPACE_ID)
air_demo(workspace_id = Sys.getenv("AIRTABLE_WORKSPACE_ID"))

## End(Not run)
```

`air_demo_setup` *Set up a demo Airtable base for exploration*

Description

Creates a new Airtable base with tables and sample records that exercise all the key field types and airtable2 features. The demo base is themed around **BollardsForArt**, a fictional creative-arts advocacy nonprofit that installs unauthorized public art, fights for arts funding, runs community workshops, and tracks their campaigns and artists.

Usage

```
air_demo_setup(
  workspace_id = Sys.getenv("AIRTABLE_WORKSPACE_ID"),
  name = "bollardsforart_demo",
  .token = NULL
)
```

Arguments

`workspace_id`	Workspace ID (defaults to AIRTABLE_WORKSPACE_ID env var). Find yours in the browser URL: `https://airtable.com/wspXXXXX/...`
`name`	Base name (default: `"bollardsforart_demo"`).
`.token`	API token (see `air_set_token()`).

Details

The demo base contains:

- **Artists** table: text, number, checkbox, single/multi-select, date, email (15+ rows with diverse international artists)
- **Projects** table: text, number, date, single-select, attachments, linked records to Artists (15+ rows with imaginative installation names)
- **Supporters** table: text, email, date – starts empty; `air_demo()` step 3 bulk-writes 120 records to demonstrate the progress bar over 12 batches

- **Grants** table: text, number, date, single-select, linked records to Projects (15+ rows of funding sources and applications)

Fields include descriptions (column metadata) set via `at_update_field()`. Project records have image attachments uploaded via `at_upload_attachment()`. Linked-records fields connect Projects to Artists and Grants to Projects.

This function creates a real Airtable base and consumes several API calls. The base cannot be deleted via the API on free/team-tier accounts; clean up manually via the Airtable web interface.

Value

The base ID (invisibly). Prints a summary of what was created.

Examples

```
## Not run:
# Requires AIRTABLE_WORKSPACE_ID and AIRTABLE_API_KEY env vars
base_id <- air_demo_setup()
air_set_base(base_id)
air_read("Artists")

## End(Not run)
```

`air_dump` *Dump an entire base (schema + data) for backup*

Description

Exports the full schema and all table data from a base. By default, attachments are downloaded to disk (`attachments = "file"`) since the purpose of a dump is to create a full backup.

Usage

```
air_dump(
  base_id,
  dir = NULL,
  format = c("list", "json", "csv"),
  attachments = c("file", "meta", "blob"),
  .token = NULL
)
```

Arguments

`base_id`	Base ID (e.g., `"appXXXXXX"`). If `NULL`, uses the session default set by `air_set_base()` or the `AIRTABLE_BASE_ID` environment variable.

dir	Directory to write files to. When format = "json" or format = "csv", schema and data files are written here. When attachments = "file", attachment files are saved under {dir}/attachments/{table_name}/{record_id}/{filename}. If NULL and format is "json" or "csv", uses a temp directory.
format	One of "list" (return as R list), "json" (write JSON files), or "csv" (write CSV files with flattened complex types).
attachments	How to handle attachment fields: "meta" (default) keeps only metadata (filename, url, size, type); "file" downloads to attachment_dir; "blob" downloads as in-memory raw vectors.
.token	Personal access token (resolved via air_token() if NULL).

Value

For format = "list": a named list with schema and a tibble per table. For format = "json" or "csv": the directory path (invisibly).

Examples

```
## Not run:
# Full backup with attachments
air_dump("appXXXXXX", dir = "backup/")

# Quick dump without downloading attachments
air_dump("appXXXXXX", dir = "backup/", attachments = "meta")

# CSV dump (flattened for spreadsheet compatibility)
air_dump("appXXXXXX", dir = "backup/", format = "csv")

## End(Not run)
```

---

air_expand_collaborator

Expand collaborator strings to list-column

Description

Inverse of air_flatten_collaborator().

Usage

```
air_expand_collaborator(x, pattern = "(.+) <(.+)>")
```

Arguments

x	A character vector (e.g., "Name <email>")
pattern	Regex with two capture groups (name, email). Default: "(.+) <(.+)>".

Value

A list of named lists with `name` and `email`.

`air_expand_multiselect` *Expand delimited strings to a multi-select list-column*

Description

Inverse of `air_flatten_multiselect()`.

Usage

```
air_expand_multiselect(x, sep = NULL)
```

Arguments

`x`	A character vector of delimited values.
`sep`	Delimiter to split on. Default `"; "` (tolerates optional surrounding spaces when the delimiter is a semicolon family), overridable via `options(airtable2.delimiter = ...)` or the `AIRTABLE2_DELIMITER` environment variable. An explicit `sep` argument always takes precedence.

Value

A list of character vectors.

`air_field_template` *Build a field template specification*

Description

Convenience function for constructing field configurations.

Usage

```
air_field_template(name, type, description = NULL, options = NULL)
```

Arguments

`name`	Field name.
`type`	Field type (e.g., `"singleLineText"`, `"number"`, `"singleSelect"`).
`description`	Optional field description.
`options`	Optional field-specific options (list). See Airtable docs for available options per type.

Value

A list suitable for use in `air_table_template()` or `at_create_field()`.

Examples

```
air_field_template("Status", "singleSelect",
  options = list(choices = list(
    list(name = "Active"), list(name = "Inactive")
  ))
)
```

`air_flatten` *Flatten a complex Airtable column to a simple atomic vector*

Description

Generic that dispatches on the `air_*` S3 class of a column, applying the appropriate per-type flattener. Plain (already-flat) vectors are returned unchanged.

Usage

```
air_flatten(x, ...)
```

Arguments

`x`	A column, typically an `air_*` list-column from `air_read()`.
`...`	Passed to the underlying flattener, e.g. `sep`, `field`, `format`.

Value

A character vector (or `x` unchanged for non-`air_*` input).

`air_flatten_attachments` *Flatten an attachments list-column to a summary string*

Description

Flatten an attachments list-column to a summary string

Usage

```
air_flatten_attachments(x, field = "filename", sep = NULL)
```

Arguments

x	A list-column where each element is a data frame or list of attachment objects.
field	Which attachment field to extract (e.g., "filename", "url"). Default "filename".
sep	Delimiter. Default "; ", overridable via options(airtable2.delimiter = ...) or the AIRTABLE2_DELIMITER environment variable. An explicit sep argument always takes precedence.

Value

A character vector.

---

air_flatten_collaborator

Flatten a collaborator list-column to strings

Description

Flatten a collaborator list-column to strings

Usage

```
air_flatten_collaborator(x, format = "{name} <{email}>")
```

Arguments

x	A list-column where each element is a named list with name and email fields.
format	A glue-style template. Available fields: id, email, name. Default: "{name} <{email}>".

Value

A character vector.

---

air_flatten_links

Flatten a record-links list-column to delimited strings

Description

Flatten a record-links list-column to delimited strings

Usage

```
air_flatten_links(x, sep = NULL)
```

Arguments

x	A list-column where each element is a character vector of record IDs.
sep	Delimiter. Default "; ", overridable via options(airtable2.delimiter = ...) or the AIRTABLE2_DELIMITER environment variable. An explicit sep argument always takes precedence.

Value

A character vector.

---

air_flatten_multiselect *Flatten a multi-select list-column to delimited strings*

Description

Flatten a multi-select list-column to delimited strings

Usage

```
air_flatten_multiselect(x, sep = NULL)
```

Arguments

x	A list-column where each element is a character vector.
sep	Delimiter to join values. Default "; ", overridable via options(airtable2.delimiter = ...) or the AIRTABLE2_DELIMITER environment variable. An explicit sep argument always takes precedence.

Value

A character vector.

---

air_left_join *Join local data with an Airtable table*

Description

Fetches a remote Airtable table and joins it with a local data frame. These are convenience wrappers around air_read() plus base-R merge(). The ... are forwarded to air_read() (e.g. formula, fields).

Usage

```
air_left_join(x, table, base_id = NULL, by = NULL, ..., .token = NULL)

air_inner_join(x, table, base_id = NULL, by = NULL, ..., .token = NULL)

air_full_join(x, table, base_id = NULL, by = NULL, ..., .token = NULL)
```

Arguments

x	A local data frame.
table	Table name or ID.
base_id	Base ID (e.g., "appXXXXXX"). If NULL, uses the session default set by air_set_base() or the AIRTABLE_BASE_ID environment variable.
by	Character vector of column name(s) to join on. If NULL, uses all column names shared between x and the remote table (excluding airtable_id and airtable_created_time).
...	Additional arguments passed to air_read().
.token	Personal access token (resolved via air_token() if NULL).

Value

A tibble.

Examples

```
## Not run:
scores <- tibble::tibble(Name = c("Alice", "Bob"), Score = c(90, 85))
air_left_join(scores, "Contacts", "appXXXX", by = "Name")

## End(Not run)
```

---

air_left_join_upload *Upload local data to matched Airtable records*

Description

air_left_join_upload() is the complement of air_left_join(). Instead of pulling remote data *into* a local data frame, it pushes the columns that a local data frame x carries *onto* matching records that already exist in an Airtable table, matching by a join key. It enriches existing records with new or changed field values.

Usage

```
air_left_join_upload(
  x,
  table,
  base_id = NULL,
  by = NULL,
  fields = NULL,
  add_fields = c("yes", "warn", "error"),
  .token = NULL
)
```

Arguments

x	A local data frame whose columns should be uploaded.
table	Table name or ID.
base_id	Base ID (e.g., "appXXXXXX"). If NULL, uses the session default set by air_set_base() or the AIRTABLE_BASE_ID environment variable.
by	The join key: a column present in both x and the remote table (e.g. "Name"), or a named character vector c(local = "remote") when the local and remote key columns differ. If NULL, uses common column names (dplyr-style) and messages which key was chosen.
fields	Optional character vector limiting which of x's columns to upload. Defaults to all columns of x except the key and the Airtable meta columns (airtable_id, airtable_created_time).
add_fields	What to do when an upload column does not exist in the table: "yes" (default) creates it, "warn" warns and drops it, "error" errors. Passed through to air_upsert().
.token	Optional API token.

Details

Matching is by by (a key present in both x and the remote table). For each matched record, only the upload columns whose value is **new** (the field does not yet exist remotely) or **changed** (differs from the current remote value) are sent, minimising API calls. Columns whose value already equals the remote value are not re-sent.

This function never inserts new records (unmatched local rows are skipped and counted) and never deletes remote fields. To insert as well as update, use air_upsert().

The remote table is read minimally: only the key field plus any to-upload fields that already exist are fetched. The actual write is delegated to air_upsert() (matching by airtable_id), so batching, throttling, API counting, and field creation are reused.

Value

Invisibly, a tibble of the records that were updated: their airtable_id, the key, and the changed field values.

See Also

air_left_join() for the read direction, air_upsert() to also insert new records.

Examples

```
## Not run:
# Local scores you computed; push them onto existing Contacts by Name.
scores <- tibble::tibble(Name = c("Alice", "Bob"), Score = c(90, 85))
air_left_join_upload(scores, "Contacts", "appXXXX", by = "Name")

# Different local/remote key names.
air_left_join_upload(scores, "Contacts", "appXXXX", by = c(person = "Name"))

## End(Not run)
```

---

air_meta *Get base metadata as a flat tibble*

Description

Returns one row per field across all tables, useful for inspecting and editing base structure as a data frame.

Usage

```
air_meta(base_id, .token = NULL)
```

Arguments

base_id	Base ID (e.g., "appXXXXXX"). If NULL, uses the session default set by air_set_base() or the AIRTABLE_BASE_ID environment variable.
.token	Personal access token (resolved via air_token() if NULL).

Value

A tibble with columns: table_name, table_id, field_name, field_id, field_type, description.

Examples

```
## Not run:
meta <- air_meta("appXXXXXX")
meta

## End(Not run)
```

air_meta_init — *Seed the _metadata table from the live schema*

Description

Reads the current base schema via `air_meta()` and upserts it into a designated table within the same base. Run this **once** to initialise the metadata store, then edit the table in Airtable and call `air_meta_sync()` to push changes back.

Usage

```
air_meta_init(base_id, meta_table = "_metadata", .token = NULL)
```

Arguments

`base_id`	Base ID (e.g., `"appXXXXXX"`). If `NULL`, uses the session default set by `air_set_base()` or the `AIRTABLE_BASE_ID` environment variable.
`meta_table`	Name of the table to store metadata in. Default `"_metadata"`.
`.token`	Personal access token (resolved via `air_token()` if `NULL`).

Value

Invisible upsert result.

Examples

```
## Not run:
# Initialise the _metadata table
air_meta_init("appXXXXXX")

# Use a custom table name
air_meta_init("appXXXXXX", meta_table = "_docs")

## End(Not run)
```

air_meta_push — *Push metadata changes back to the base*

Description

Compares a modified metadata tibble (from `air_meta()`) against the current schema and applies name/description changes via PATCH. Changes to `table_name` rename the table; changes to `field_name` or `description` rename or re-describe the field.

Usage

```
air_meta_push(base_id, meta, .token = NULL)
```

Arguments

base_id	Base ID (e.g., "appXXXXXX"). If NULL, uses the session default set by air_set_base() or the AIRTABLE_BASE_ID environment variable.
meta	A tibble from air_meta() with modifications to table_name, field_name, or description.
.token	Personal access token (resolved via air_token() if NULL).

Value

Invisible NULL. Side effect: updates table and field metadata.

---

air_meta_sync *Sync a metadata source to patch the base schema*

Description

The **preferred metadata workflow**: pull field names and descriptions from a source (default: the "_metadata" table inside the same base), compare against the live schema, and PATCH any changed fields.

Usage

```
air_meta_sync(base_id, source = "_metadata", ..., .token = NULL)
```

Arguments

base_id	Base ID (e.g., "appXXXXXX"). If NULL, uses the session default set by air_set_base() or the AIRTABLE_BASE_ID environment variable.
source	Where to pull the edited metadata from. Defaults to "_metadata", meaning the _metadata table inside base_id. Can also be a data.frame, a path to a .csv file, or a path to a .json file (see *Source precedence* above).
...	Ignored; forces .token to be named.
.token	Personal access token (resolved via air_token() if NULL).

Value

Invisible NULL. Side-effect: PATCHes changed fields in the base.

Typical workflow

1. **Seed** – run air_meta_init() once to create / populate the "_metadata" table from the live schema.
2. **Edit** – open the "_metadata" table in Airtable and change field_name / description cells to your liking.
3. **Sync** – call air_meta_sync() (no arguments needed) to PATCH the schema with your edits.

Source precedence

`source` is resolved in this order:

1. A `data.frame` → used directly.
2. A length-1 character path to an **existing** `.csv` file → read with `utils::read.csv()`.
3. A length-1 character path to an **existing** `.json` file → read with `jsonlite::fromJSON()`.
4. A length-1 character string that matches neither 2 nor 3 → treated as a **table name** inside the base and read with `air_read()`.

The source tibble must contain at least `field_id`, `table_id`, `field_name`, and `description` columns (the shape produced by `air_meta()` and `air_meta_init()`). An extra `meta_key` column (produced by older seed runs) is silently ignored.

Examples

```
## Not run:
# 1. Seed the _metadata table from the live schema (run once)
air_meta_init("appXXXXXX")

# 2. Edit field_name / description cells directly in Airtable ...

# 3. Pull edits back and patch the schema
air_meta_sync("appXXXXXX")

# 4. Alternatively, sync from a local CSV
air_meta_sync("appXXXXXX", source = "my_meta.csv")

## End(Not run)
```

`air_pane` *Open the Airtable Connection Pane*

Description

Establishes a connection via `air_connect()` and ensures it is registered with the RStudio/Positron connection pane.

Usage

```
air_pane(base = NULL, bases = NULL, .token = NULL, include_views = FALSE)
```

Arguments

`base`	Character. A Base ID (starts with `app`) or a Base Name. If `NULL`, connects to all accessible bases.
`bases`	Character vector of Base IDs or names. When supplied, the connection pane shows only those bases (as schemas). Cannot be combined with `base`.

.token	Character. Airtable Personal Access Token (resolved via air_token() if NULL).
include_views	Logical. If TRUE, views are included in the connection pane alongside tables. Default FALSE.

Value

An AirtableConnection object (invisibly).

---

air_read *Read records from an Airtable table*

Description

High-level function that auto-paginates, fetches schema for type coercion, and returns a properly typed tibble.

Usage

```
air_read(
  table,
  base_id = NULL,
  view = NULL,
  fields = NULL,
  formula = NULL,
  sort = NULL,
  max_records = Inf,
  page_size = 100L,
  coerce = TRUE,
  attachments = c("meta", "file", "blob"),
  attachment_dir = NULL,
  parallel = NULL,
  progress = NULL,
  .token = NULL
)
```

Arguments

table	Table name or ID.
base_id	Base ID (e.g., "appXXXXXX"). If NULL, uses the session default set by air_set_base() or the AIRTABLE_BASE_ID environment variable.
view	Optional view name or ID to filter by.
fields	Optional character vector of field names to return.
formula	Optional Airtable formula string for filtering.
sort	Optional named character vector for sorting (names = field names, values = "asc" or "desc").

max_records	Maximum number of records to return. Default Inf (all).
page_size	Records per page (max 100).
coerce	If TRUE (default), fetches schema and coerces types. If FALSE, returns raw values (faster but untyped).
attachments	How to handle attachment fields: "meta" (default) keeps only metadata (filename, url, size, type); "file" downloads to attachment_dir; "blob" downloads as in-memory raw vectors.
attachment_dir	Directory for downloading attachments (required when attachments = "file"). Files are saved as {attachment_dir}/{record_id}/{filename}.
parallel	Logical or NULL. If TRUE, attachment downloads use httr2::req_perform_parallel() (up to 5 concurrent). If NULL, uses option airtable2.parallel or env var AIRTABLE2_PARALLEL (default TRUE).
progress	Logical or NULL. If TRUE, shows a cli progress bar for paginated requests. If NULL (default), uses option airtable2.progress.bar or env var AIRTABLE2_PROGRESS_BAR.
.token	Personal access token (resolved via air_token() if NULL).

Value

A tibble with columns airtable_id, airtable_created_time, and one column per field.

Examples

```
## Not run:
# Read all records from a table
df <- air_read("Contacts", "appXXXXXX")

# Read specific fields with a filter
df <- air_read("Contacts", "appXXXXXX",
  fields = c("Name", "Email"),
  formula = "{Age} > 30"
)

# Download attachments to disk
df <- air_read("Projects", "appXXXXXX",
  attachments = "file",
  attachment_dir = "downloads/"
)

## End(Not run)
```

---

air_read_attachments *Read attachments from records*

Description

Downloads attachment files from a specified field, returning them as either in-memory blobs or saved files.

Usage

```
air_read_attachments(
  base_id,
  table,
  field,
  record_ids = NULL,
  dest = c("blob", "file"),
  dir = NULL,
  parallel = NULL,
  .token = NULL
)
```

Arguments

base_id	Base ID (e.g., "appXXXXXX"). If NULL, uses the session default set by air_set_base() or the AIRTABLE_BASE_ID environment variable.
table	Table name or ID.
field	Name of the attachment field.
record_ids	Optional character vector of specific record IDs to fetch. If NULL, fetches all records.
dest	Either "blob" (return raw content as list-column) or "file" (save to disk).
dir	Directory to save files to (required if dest = "file").
parallel	Logical or NULL. If TRUE, attachment downloads use httr2::req_perform_parallel() (up to 5 concurrent). If NULL, uses option airtable2.parallel or env var AIRTABLE2_PARALLEL (default TRUE).
.token	Personal access token (resolved via air_token() if NULL).

Value

A tibble with airtable_id, filename, url, and either blob (raw list-column) or local_path (character).

---

air_req *Build an httr2 request to the Airtable API*

Description

Constructs a base request with auth, user-agent, retry policy, and throttle.

Usage

```
air_req(endpoint, token = NULL, host = c("api", "content"))
```

Arguments

endpoint	Path appended to the host root (e.g., "meta/bases" or "appXXX/TableName").
token	Personal access token (resolved via air_token() if NULL).
host	Which Airtable host to target: "api" (the default REST API at https://api.airtable.com/v0) or "content" (the attachment-upload host at https://content.airtable.com/v0).

Value

An httr2_request object ready for further modification.

---

air_resolve_id *Resolve an Airtable ID or URL to its component parts*

Description

Takes an ID string (e.g., "appXXXXXX", "tblXXXXXX", "wspXXXXXX", "viwXXXXXX") or a full URL and extracts the ID, determining its type.

Usage

```
air_resolve_id(x)
```

Arguments

x	Character string. An ID, a connection object, or a URL.

Value

A list with components: type ("workspace", "base", "table", "view", or "record"), id (the extracted ID), and any other parsed components.

See Also

air_browse() for opening the ID in a browser.

Examples

```
air_resolve_id("appXXXXXX")
air_resolve_id("wspXXXXXX")
air_resolve_id("https://airtable.com/appXXXXXX/tblXXXXXX/viwXXXXXX")
```

`air_restore` *Restore a base from a dump*

Description

Recreates a base from output of `air_dump()`. When `attachments` is `"file"`, uploads attachment files from the dump directory after record creation. For CSV dumps, automatically detects and parses the flattened format.

Usage

```
air_restore(
  dump,
  base_name = NULL,
  workspace_id = NULL,
  attachments = c("file", "meta"),
  attachment_dir = NULL,
  restore_linked_fields = TRUE,
  .token = NULL
)
```

Arguments

`dump`	Either a list (from `air_dump(format = "list")`) or a path to a dump directory (from `air_dump(format = "json")` or `air_dump(format = "csv")`).
`base_name`	Name for the new base. If `NULL`, uses a generated name.
`workspace_id`	Workspace ID to create the base in.
`attachments`	How to handle attachment fields: `"meta"` (default) keeps only metadata (filename, url, size, type); `"file"` downloads to `attachment_dir`; `"blob"` downloads as in-memory raw vectors.
`attachment_dir`	Directory for downloading attachments (required when `attachments = "file"`). Files are saved as `{attachment_dir}/{record_id}/{filename}`.
`restore_linked_fields`	If `TRUE` (the default), after all records are created, linked-record fields (`multipleRecordLinks`) and their dependent computed fields (`rollup`, `lookup`, `count`) are recreated with remapped table/field IDs, and the link cell values (record-to-record connections) are repopulated by remapping old record IDs to the newly created record IDs. Set to `FALSE` to skip this step (faster, but links will be empty).
`.token`	Personal access token (resolved via `air_token()` if `NULL`).

Value

The new base ID (invisibly).

Linked-record fields

Linked-record fields (`multipleRecordLinks`) and the computed fields that depend on them (`rollup`, `lookup`, `count`) are skipped during the initial field-creation pass because their options reference base-specific IDs. When `restore_linked_fields = TRUE` (the default), after all records are inserted a two-step pass runs:

1. **Field definitions**: `multipleRecordLinks` fields are recreated with `linkedTableId` remapped to the new base's table IDs.
2. **Cell values**: the link columns in the dump contain old record IDs. These are remapped to the new record IDs (matched by insertion order) and written back via `air_upsert()`.

Examples

```
## Not run:
# Restore from a directory dump
air_restore("backup/", workspace_id = "wspXXXXXX")

# Restore from a CSV dump
air_restore("backup/", workspace_id = "wspXXXXXX", format = "csv")

## End(Not run)
```

`air_schema` *Get schema for a base as a tidy tibble*

Description

Returns table and field metadata in a structured tibble format.

Usage

```
air_schema(base_id, .token = NULL)
```

Arguments

`base_id`	Base ID (e.g., `"appXXXXXX"`). If `NULL`, uses the session default set by `air_set_base()` or the `AIRTABLE_BASE_ID` environment variable.
`.token`	Personal access token (resolved via `air_token()` if `NULL`).

Value

A tibble with columns `table_id`, `table_name`, and `fields` (a list-column of tibbles with `id`, `name`, `type`, `description`).

Examples

```
## Not run:
schema <- air_schema("appXXXXXX")
schema$table_name
schema$fields[[1]]

## End(Not run)
```

`air_set_base` *Set the default Airtable base for this session*

Description

Validates that `base_id` is accessible with the current token, prints a `{cli}` confirmation, and sets `options(airtable2.base_id = base_id)` for the rest of the session. Functions that accept `base_id` will use this default when `base_id` is `NULL`.

Usage

```
air_set_base(base_id, .token = NULL)
```

Arguments

`base_id`	Base ID (starts with app).
`.token`	Personal access token (resolved via `air_token()` if `NULL`).

Value

`base_id` (invisibly).

Examples

```
## Not run:
air_set_base("appXXXXXXXXXXXXXX")

## End(Not run)
```

air_set_token *Set the default Airtable token for this session*

Description

Validates the token by calling `at_whoami()`, prints a `{cli}` confirmation with the authenticated user's email, and sets `options(airtable2.token = tok)`.

Usage

```
air_set_token(tok)
```

Arguments

`tok`	Personal Access Token (PAT) string.

Value

`tok` (invisibly).

Examples

```
## Not run:
air_set_token(Sys.getenv("AIRTABLE_API_KEY"))

## End(Not run)
```

air_simplify *Simplify all complex columns in a tibble for display/export*

Description

Applies the appropriate flatten function to each list-column based on schema information.

Usage

```
air_simplify(data, schema = NULL)
```

Arguments

`data`	A tibble (typically from `air_read()`).
`schema`	Optional list of field definitions (from `at_get_schema()`). If `NULL`, uses heuristics.

Value

A tibble with list-columns replaced by character representations.

air_sync *Smart sync: diff-based upsert + delete*

Description

Compares local data against the current table contents using a hash of specified fields, then creates/updates/deletes as needed.

Usage

```
air_sync(
  data,
  table,
  key,
  base_id = NULL,
  hash_fields = NULL,
  delete_missing = TRUE,
  typecast = TRUE,
  add_fields = c("error", "warn", "yes"),
  attachments = c("meta", "file", "blob"),
  attachment_dir = NULL,
  progress = NULL,
  .token = NULL
)
```

Arguments

data	A data frame representing the desired state of the table. May contain computed field columns (they are ignored).
table	Table name or ID.
key	Column name in data that uniquely identifies records (used as the merge field for upsert). Must be a single field.
base_id	Base ID (e.g., "appXXXXXX"). If NULL, uses the session default set by air_set_base() or the AIRTABLE_BASE_ID environment variable.
hash_fields	Character vector of fields to include in the change- detection hash. If NULL (default), all non-key, non-computed, non-attachment fields are used. Computed and attachment fields are always excluded even if explicitly listed.
delete_missing	If TRUE (default), records in Airtable that are not in data will be deleted.
typecast	If TRUE (default), Airtable will attempt to coerce values.
add_fields	What to do when data contains columns not in the table. Passed to air_upsert(). Default is "error".
attachments	How to handle attachment fields: "meta" (default) keeps only metadata (filename, url, size, type); "file" downloads to attachment_dir; "blob" downloads as in-memory raw vectors.

attachment_dir	Directory for downloading attachments (required when attachments = "file"). Files are saved as {attachment_dir}/{record_id}/{filename}.
progress	Logical or NULL. If TRUE, shows a cli progress bar for read and upsert operations. If NULL (default), uses option airtable2.progress.bar or env var AIRTABLE2_PROGRESS_BAR.
.token	Personal access token (resolved via air_token() if NULL).

Details

Computed fields (formulas, rollups, autoNumber, createdTime, lastModifiedTime, createdBy, lastModifiedBy, etc.) are automatically:

- Excluded from the change-detection hash (they change server-side and would cause spurious "changed" detections on every sync)
- Excluded from the upload payload (the API rejects writes to them)

Attachment fields (multipleAttachments) are always excluded from the change-detection hash because their URLs are volatile (expire hourly). When attachments is "file" or "blob", attachment content is uploaded for newly created records after the sync completes.

Value

A list with counts: created, updated, deleted, unchanged (invisibly).

Examples

```
## Not run:
desired <- data.frame(Name = c("Alice", "Bob"), Age = c(30, 26))
result <- air_sync(desired, "Contacts", key = "Name", base_id = "appXXXXXX")
result$created
result$unchanged

## End(Not run)
```

---

air_sync_attachments *Smart sync attachments*

Description

Compares filenames between local data and remote records, uploads new/changed files, skips unchanged.

Usage

```
air_sync_attachments(
  base_id,
  table,
  field,
  data,
  key,
  parallel = NULL,
  .token = NULL
)
```

Arguments

base_id	Base ID (e.g., "appXXXXXX"). If NULL, uses the session default set by air_set_base() or the AIRTABLE_BASE_ID environment variable.
table	Table name or ID.
field	Name of the attachment field.
data	A tibble with a key column and file_path column.
key	Column name that identifies which record to attach to.
parallel	Logical or NULL. If TRUE, attachment downloads use httr2::req_perform_parallel() (up to 5 concurrent). If NULL, uses option airtable2.parallel or env var AIRTABLE2_PARALLEL (default TRUE).
.token	Personal access token (resolved via air_token() if NULL).

Value

A list with counts: uploaded, skipped (invisibly).

---

air_table_template *Build a table template specification*

Description

Convenience function for constructing table configurations suitable for at_create_table() or at_create_base().

Usage

```
air_table_template(name, fields, description = NULL)
```

Arguments

name	Table name.
fields	A list of field specifications (e.g., from air_field_template()).
description	Optional table description.

Value

A list suitable for passing to the Airtable API.

Examples

```
fields <- list(
  air_field_template("Name", "singleLineText"),
  air_field_template("Score", "number", options = list(precision = 2))
)
air_table_template("Results", fields, description = "Exam results")
```

---

air_token *Resolve an Airtable personal access token*

Description

Looks for a token in this order:

1. Explicit `.token` argument
2. `getOption("airtable2.token")`
3. `Sys.getenv("AIRTABLE_API_KEY")`

Usage

```
air_token(token = NULL)
```

Arguments

`token`	A personal access token string, or `NULL` to use defaults.

Value

A string (the token).

Examples

```
## Not run:
# Uses AIRTABLE_API_KEY env var by default
token <- air_token()

# Or pass explicitly
token <- air_token("patXXXXXXXX")

## End(Not run)
```

`air_upsert` *Upsert records into an Airtable table*

Description

Uses Airtable's native upsert (PATCH with `performUpsert`) to create or update records based on merge fields. Supports two matching modes:

Usage

```
air_upsert(
  data,
  table,
  merge_on,
  base_id = NULL,
  typecast = TRUE,
  add_fields = c("error", "warn", "yes"),
  attachments = c("meta", "file", "blob"),
  attachment_dir = NULL,
  progress = NULL,
  .token = NULL
)
```

Arguments

`data`	A data frame of records to upsert. May include an `airtable_id` column for direct record matching. Computed field columns and attachment field columns are silently dropped from the record payload.
`table`	Table name or ID.
`merge_on`	Character vector of 1-3 field names to match on (for records without an `airtable_id`).
`base_id`	Base ID (e.g., `"appXXXXXX"`). If `NULL`, uses the session default set by `air_set_base()` or the `AIRTABLE_BASE_ID` environment variable.
`typecast`	If `TRUE` (default), Airtable will attempt to coerce values.
`add_fields`	What to do when `data` contains columns not in the table: • `"error"` (default): error if unknown columns exist. • `"warn"`: warn and drop unknown columns. • `"yes"`: create missing fields before upserting (as `singleLineText`).
`attachments`	How to handle attachment fields: `"meta"` (default) keeps only metadata (filename, url, size, type); `"file"` downloads to `attachment_dir`; `"blob"` downloads as in-memory raw vectors.
`attachment_dir`	Directory for downloading attachments (required when `attachments = "file"`). Files are saved as `{attachment_dir}/{record_id}/{filename}`.
`progress`	Logical or `NULL`. If `TRUE`, shows a cli progress bar for batch operations. If `NULL` (default), uses option `airtable2.progress.bar` or env var `AIRTABLE2_PROGRESS_BAR`.
`.token`	Personal access token (resolved via `air_token()` if `NULL`).

Details

- If `data` contains an `airtable_id` column, records with non-NA IDs are updated directly by record ID (more efficient).
- Records without an `airtable_id` (or where it is `NA`) are matched using the `merge_on` field(s) via Airtable's upsert mechanism.

Computed fields (formulas, rollups, autoNumber, createdTime, lastModifiedTime, etc.) and attachment fields are automatically excluded from the upload payload. When `attachments` is `"file"` or `"blob"`, attachment content is uploaded separately after record creation/update. Optionally creates missing columns.

Value

A list with `created` and `updated` character vectors of record IDs (invisibly).

Examples

```
## Not run:
data <- data.frame(Name = c("Alice", "Bob"), Age = c(31, 26))
result <- air_upsert(data, "Contacts", merge_on = "Name", base_id = "appXXXXXX")
result$created
result$updated

## End(Not run)
```

`air_write` *Write (create) records in an Airtable table*

Description

Converts a data frame into records and creates them in the specified table. Automatically batches in groups of 10. Computed fields (formulas, rollups, autoNumber, createdTime, lastModifiedTime, etc.) and attachment fields are automatically excluded from the upload payload. When `attachments` is `"file"` or `"blob"`, attachment content is uploaded separately after record creation using the dedicated upload endpoint.

Usage

```
air_write(
  data,
  table,
  base_id = NULL,
  typecast = TRUE,
  add_fields = c("error", "warn", "yes"),
  attachments = c("meta", "file", "blob"),
  attachment_dir = NULL,
  progress = NULL,
  .token = NULL
)
```

Arguments

data	A data frame of records to create. Should not contain airtable_id (those would be ignored). Computed field columns and attachment field columns are silently dropped from the record payload.
table	Table name or ID.
base_id	Base ID (e.g., "appXXXXXX"). If NULL, uses the session default set by air_set_base() or the AIRTABLE_BASE_ID environment variable.
typecast	If TRUE (default), Airtable will attempt to coerce values to match field types.
add_fields	What to do when data contains columns not in the table: • "error" (default): error if unknown columns exist. • "warn": warn and drop unknown columns. • "yes": create missing fields before writing (as singleLineText).
attachments	How to handle attachment fields: "meta" (default) keeps only metadata (filename, url, size, type); "file" downloads to attachment_dir; "blob" downloads as in-memory raw vectors.
attachment_dir	Directory for downloading attachments (required when attachments = "file"). Files are saved as {attachment_dir}/{record_id}/{filename}.
progress	Logical or NULL. If TRUE, shows a cli progress bar for paginated requests. If NULL (default), uses option airtable2.progress.bar or env var AIRTABLE2_PROGRESS_BAR.
.token	Personal access token (resolved via air_token() if NULL).

Value

A character vector of the created record IDs (invisibly).

Examples

```
## Not run:
data <- data.frame(Name = c("Alice", "Bob"), Age = c(30, 25))
ids <- air_write(data, "Contacts", "appXXXXXX")

# Write records and upload attachments from list-column
ids <- air_write(data, "Projects", "appXXXXXX",
  attachments = "file",
  attachment_dir = "files/"
)

# Write records and add new columns if they don't exist
ids <- air_write(data, "Contacts", "appXXXXXX", add_fields = "yes")

## End(Not run)
```

air_write_attachments *Upload attachments to records*

Description

Upload attachments to records

Usage

```
air_write_attachments(
  base_id,
  table,
  field,
  data,
  parallel = NULL,
  .token = NULL
)
```

Arguments

base_id	Base ID (e.g., "appXXXXXX"). If NULL, uses the session default set by air_set_base() or the AIRTABLE_BASE_ID environment variable.
table	Table name or ID.
field	Name of the attachment field.
data	A tibble with airtable_id and file_path columns.
parallel	Logical or NULL. If TRUE, attachment downloads use httr2::req_perform_parallel() (up to 5 concurrent). If NULL, uses option airtable2.parallel or env var AIRTABLE2_PARALLEL (default TRUE).
.token	Personal access token (resolved via air_token() if NULL).

Value

Invisible NULL. Side effect: uploads attachments.

airtable2 *Airtable DBI driver*

Description

Create a DBI driver for Airtable. Use with DBI::dbConnect() to create an Airtable DBI connection.

Usage

```
airtable2()

at()

airtable()

## S4 method for signature 'AirtableDriver'
dbConnect(
  drv,
  token = NULL,
  base_id = NULL,
  include_views = FALSE,
  connect_code = NULL,
  bases_filter = NULL,
  ...
)

## S4 method for signature 'AirtableDriver'
dbDataType(dbObj, obj, ...)

## S4 method for signature 'AirtableDriver'
dbUnloadDriver(drv, ...)
```

Arguments

drv, dbObj	An AirtableDriver object.
token	Personal access token (resolved via air_token() if NULL).
base_id	Optional Airtable base ID. If omitted, all accessible bases are shown as schemas in the connection pane.
include_views	Logical. If TRUE, views are listed in the connection pane as separate objects alongside tables.
connect_code	Optional custom reconnect code string for the IDE connection pane. If NULL (default), a code snippet is auto-generated.
bases_filter	Optional character vector of base IDs or names to restrict the connection pane to specific bases.
...	Additional arguments passed to DBI methods.
obj	An R object to map to an Airtable field type.

Value

An AirtableDriver object.

Usage

Use this driver to create DBI-compliant connections to Airtable for use with RStudio/Positron's connection pane. See airtable2-package for package-level documentation and AirtableConnection

for details on available DBI methods.

Examples

```
## Not run:
con <- DBI::dbConnect(airtable2(), base_id = "appXXXXXX")
DBI::dbListTables(con)
DBI::dbDisconnect(con)

## End(Not run)
```

`AirtableConnection-class` *Airtable DBI connection*

Description

Stores Airtable credentials and connection state for DBI methods. Use `airtable2()` with `DBI::dbConnect()` to create connections.

Usage

```
## S4 method for signature 'AirtableConnection'
dbDisconnect(conn, ...)

## S4 method for signature 'AirtableConnection'
dbIsValid(dbObj, ...)

## S4 method for signature 'AirtableConnection'
dbListTables(conn, ...)

## S4 method for signature 'AirtableConnection,character'
dbExistsTable(conn, name, ...)

## S4 method for signature 'AirtableConnection,character'
dbListFields(conn, name, ...)

## S4 method for signature 'AirtableConnection,character'
dbReadTable(conn, name, ...)

## S4 method for signature 'AirtableConnection,character'
dbWriteTable(conn, name, value, overwrite = FALSE, append = FALSE, ...)

## S4 method for signature 'AirtableConnection,character'
dbRemoveTable(conn, name, ...)

## S4 method for signature 'AirtableConnection'
dbGetInfo(dbObj, ...)
```

Arguments

conn, dbObj	An AirtableConnection object.
...	Additional arguments passed to Airtable helpers.
name	Table name.
value	Data frame to write.
overwrite, append	DBI write mode flags.

Capabilities and limitations

Reading tables dbReadTable() works on any accessible table. You can also pass "TableName WHERE <formula>" as the name to filter records using Airtable's formula syntax.

Writing tables dbWriteTable() works on **existing** tables only. With append = TRUE it creates new records; with overwrite = TRUE it syncs (upsert + delete-missing) using the first column as the key.

No table creation via DBI Use at_create_table() to create tables. dbWriteTable() errors if the table does not already exist.

No table removal Airtable's API cannot delete tables. Use the Airtable web UI instead. dbRemoveTable() will error with a clear message.

No SQL queries Arbitrary SQL is not supported. Use the high-level helpers (air_read(), air_write(), air_upsert(), air_sync()) for more ergonomic access.

No transactions Airtable does not support database transactions.

Single-base or all-bases When a base_id is given, the connection shows that base's tables directly. Without a base_id, all accessible bases appear as schemas in the connection pane.

For most use cases, the high-level functions like air_read(), air_write(), air_upsert(), and air_sync() provide more ergonomic interfaces for Airtable operations.

AirtableDriver-class *Airtable DBI driver class*

Description

S4 class representing the Airtable DBI driver. Use airtable2() to construct an instance and pass it to DBI::dbConnect().

`AirtableResult-class` *Airtable DBI result*

Description

Eagerly materialized DBI result object used by `DBI::dbSendQuery()`. Unlike traditional DBI results which may be cursor-based, AirtableResult objects fetch all matching records immediately (due to Airtable API limitations).

Usage

```
## S4 method for signature 'AirtableConnection,character'
dbSendQuery(conn, statement, ...)

## S4 method for signature 'AirtableResult,numeric'
dbFetch(res, n = -1, ...)

## S4 method for signature 'AirtableResult'
dbClearResult(res, ...)

## S4 method for signature 'AirtableResult'
dbHasCompleted(res, ...)

## S4 method for signature 'AirtableResult'
dbGetRowCount(res, ...)

## S4 method for signature 'AirtableResult'
dbGetStatement(res, ...)

## S4 method for signature 'AirtableResult'
dbIsValid(dbObj, ...)

## S4 method for signature 'AirtableResult'
dbGetRowsAffected(res, ...)

## S4 method for signature 'AirtableResult,missing'
dbFetch(res, n = -1, ...)
```

Arguments

`conn`	An `AirtableConnection` object.
`statement`	Table name, optionally followed by `WHERE <formula>`.
`...`	Additional arguments passed to Airtable helpers.
`res, dbObj`	An `AirtableResult` object.
`n`	Number of rows to fetch. Use a negative value to fetch all rows. Note: Due to eager evaluation, this parameter is primarily for compatibility.

Limitations

Eager evaluation All data is fetched when the result is created. There is no cursor-based iteration.

Read-only AirtableResult objects are for reading data only.

Formula filtering The `statement` parameter supports Airtable formula syntax after the table name (e.g., `"Contacts WHERE Age > 30"`). This is not SQL.

`at_create_base` *Create a new base*

Description

Create a new base

Usage

```
at_create_base(name, tables, workspace_id = NULL, token = NULL)
```

Arguments

`name`	Name for the new base.
`tables`	A list of table configurations. Each should include at minimum `name` and `fields` (a list of field configs).
`workspace_id`	Workspace ID to create the base in.
`token`	Personal access token (resolved via `air_token()` if `NULL`).

Value

The created base object (list with `id`, `name`, `tables`).

Examples

```
## Not run:
new_base <- at_create_base(
  name = "My New Base",
  workspace_id = "wspXXXXXXXXXXXXX",
  tables = list(list(
    name = "Items",
    fields = list(list(name = "Name", type = "singleLineText"))
  ))
)
new_base$id

## End(Not run)
```

`at_create_field` *Create a field in a table*

Description

Create a field in a table

Usage

```
at_create_field(
  name,
  table_id,
  type,
  base_id = NULL,
  description = NULL,
  options = NULL,
  token = NULL
)
```

Arguments

`name`	Field name.
`table_id`	Table ID.
`type`	Field type (e.g., `"singleLineText"`, `"number"`).
`base_id`	Base ID. If `NULL`, uses the session default set by `air_set_base()` or the `AIRTABLE_BASE_ID` environment variable.
`description`	Optional field description.
`options`	Field-specific options (list). Depends on field type.
`token`	Personal access token (resolved via `air_token()` if `NULL`).

Value

The created field object.

Examples

```
## Not run:
# Add a number field
at_create_field(
  name     = "Score",
  table_id = "tblXXXXXXXXXXXXX",
  type     = "number",
  base_id  = "appXXXXXXXXXXXXX",
  options  = list(precision = 1L)
)
```

```
# Add a single-select field
at_create_field(
  name     = "Status",
  table_id = "tblXXXXXXXXXXXXX",
  type     = "singleSelect",
  base_id  = "appXXXXXXXXXXXXX",
  options  = list(choices = list(list(name = "Open"), list(name = "Closed")))
)

## End(Not run)
```

`at_create_records` *Create records in a table*

Description

Creates up to 10 records per API call. If more than 10 records are provided, they are automatically batched.

Usage

```
at_create_records(
  base_id,
  table_id,
  records,
  typecast = FALSE,
  token = NULL,
  progress = NULL
)
```

Arguments

`base_id`	Base ID (e.g., `"appXXXXXX"`).
`table_id`	Table name or ID.
`records`	A list of record objects. Each should be a list with a `fields` element (a named list of field values).
`typecast`	If `TRUE`, Airtable will attempt to cast values to the correct type.
`token`	Personal access token (resolved via `air_token()` if `NULL`).
`progress`	Logical or `NULL`. If `TRUE`, shows a cli progress bar for batch operations. If `NULL` (default), uses option `airtable2.progress.bar` or env var `AIRTABLE2_PROGRESS_BAR` (both default to `FALSE`).

Value

A list of created record objects (with assigned IDs).

Examples

```
## Not run:
records <- list(
  list(fields = list(Name = "Alice", Age = 30)),
  list(fields = list(Name = "Bob",   Age = 25))
)
created <- at_create_records("appXXXXXXXXXXXXXX", "Contacts", records)
vapply(created, function(r) r$id, character(1))

## End(Not run)
```

`at_create_table` *Create a table in a base*

Description

Create a table in a base

Usage

```
at_create_table(name, fields, base_id = NULL, description = NULL, token = NULL)
```

Arguments

`name`	Table name.
`fields`	A list of field configurations. The first field becomes the primary field.
`base_id`	Base ID. If `NULL`, uses the session default set by `air_set_base()` or the `AIRTABLE_BASE_ID` environment variable.
`description`	Optional table description.
`token`	Personal access token (resolved via `air_token()` if `NULL`).

Value

The created table object (list with `id`, `name`, `fields`).

Examples

```
## Not run:
tbl <- at_create_table(
  name = "Tasks",
  fields = list(
    list(name = "Title",  type = "singleLineText"),
    list(name = "Done",   type = "checkbox",
         options = list(icon = "check", color = "greenBright")),
    list(name = "Due",    type = "date",
         options = list(dateFormat = list(name = "iso")))
  ),
  base_id = "appXXXXXXXXXXXXX"
```

```
)
tbl$id

## End(Not run)
```

`at_delete_records` *Delete records*

Description

Deletes up to 10 records per API call. Automatically batched.

Usage

```
at_delete_records(base_id, table_id, record_ids, token = NULL, progress = NULL)
```

Arguments

`base_id`	Base ID (e.g., `"appXXXXXX"`).
`table_id`	Table name or ID.
`record_ids`	Character vector of record IDs to delete.
`token`	Personal access token (resolved via `air_token()` if `NULL`).
`progress`	Logical or `NULL`. If `TRUE`, shows a cli progress bar for batch operations. If `NULL` (default), uses option `airtable2.progress.bar` or env var `AIRTABLE2_PROGRESS_BAR`.

Value

A list of delete confirmation objects (each with `id` and `deleted = TRUE`).

Examples

```
## Not run:
at_delete_records(
  "appXXXXXXXXXXXXXX",
  "Contacts",
  c("recXXXXXXXXXXXXX", "recYYYYYYYYYYYYY")
)

## End(Not run)
```

at_get_base *Get information about a single base*

Description

Get information about a single base

Usage

```
at_get_base(base_id, token = NULL)
```

Arguments

base_id	Base ID (e.g., "appXXXXXX").
token	Personal access token (resolved via air_token() if NULL).

Value

A list with id, name, and permissionLevel, or NULL if not found.

Examples

```
## Not run:
info <- at_get_base("appXXXXXXXXXXXXXX")
info$name
info$permissionLevel

## End(Not run)
```

at_get_collaborators *Get collaborators for a base*

Description

Get collaborators for a base

Usage

```
at_get_collaborators(base_id, token = NULL)
```

Arguments

base_id	Base ID (e.g., "appXXXXXX").
token	Personal access token (resolved via air_token() if NULL).

Value

The parsed collaborator response (list).

Examples

```
## Not run:
collabs <- at_get_collaborators("appXXXXXXXXXXXXXX")

## End(Not run)
```

`at_get_record` *Get a single record*

Description

Get a single record

Usage

```
at_get_record(base_id, table_id, record_id, token = NULL)
```

Arguments

`base_id`	Base ID (e.g., `"appXXXXXX"`).
`table_id`	Table name or ID.
`record_id`	Record ID (e.g., `"recXXXXXX"`).
`token`	Personal access token (resolved via `air_token()` if `NULL`).

Value

A single record object (list with `id`, `createdTime`, `fields`).

Examples

```
## Not run:
rec <- at_get_record(
  "appXXXXXXXXXXXXXX",
  "Contacts",
  "recXXXXXXXXXXXXX"
)
rec$fields$Name

## End(Not run)
```

at_get_schema *Get the schema (tables + fields) for a base*

Description

Get the schema (tables + fields) for a base

Usage

```
at_get_schema(base_id, token = NULL)
```

Arguments

base_id	Base ID (e.g., "appXXXXXX").
token	Personal access token (resolved via air_token() if NULL).

Value

A list of table objects, each containing id, name, description, fields, and views.

Examples

```
## Not run:
tables <- at_get_schema("appXXXXXXXXXXXXXX")
vapply(tables, function(t) t$name, character(1))

## End(Not run)
```

at_get_view *Get a specific view's metadata*

Description

Get a specific view's metadata

Usage

```
at_get_view(base_id, table_id, view_id, token = NULL)
```

Arguments

base_id	Base ID.
table_id	Table ID.
view_id	View ID.
token	Personal access token (resolved via air_token() if NULL).

Value

A list with view metadata (name, type, formula, filterByFormula, etc.).

Examples

```
## Not run:
view <- at_get_view(
  "appXXXXXXXXXXXXXX",
  "tblXXXXXXXXXXXXX",
  "viwXXXXXXXXXXXXX"
)
view$name
view$type

## End(Not run)
```

`at_list_bases` *List all accessible bases*

Description

List all accessible bases

Usage

```
at_list_bases(token = NULL)
```

Arguments

`token`	Personal access token (resolved via `air_token()` if `NULL`).

Value

A tibble with columns `id`, `name`, and `permissionLevel`.

Examples

```
## Not run:
# List all bases accessible with the current token
at_list_bases()

## End(Not run)
```

`at_list_records` *List records from a table*

Description

List records from a table

Usage

```
at_list_records(
  base_id,
  table_id,
  fields = NULL,
  formula = NULL,
  sort = NULL,
  view = NULL,
  max_records = Inf,
  page_size = 100L,
  cell_format = NULL,
  time_zone = NULL,
  user_locale = NULL,
  return_fields_by_id = FALSE,
  token = NULL,
  progress = NULL
)
```

Arguments

`base_id`	Base ID (e.g., `"appXXXXXX"`).
`table_id`	Table name or ID.
`fields`	Character vector of field names to return.
`formula`	Airtable formula string for filtering.
`sort`	A named character vector: names are field names, values are `"asc"` or `"desc"`.
`view`	View name or ID to filter by.
`max_records`	Maximum total records to return.
`page_size`	Records per page (max 100).
`cell_format`	Either `"json"` (default) or `"string"`.
`time_zone`	Time zone for date formatting (when `cell_format = "string"`).
`user_locale`	Locale for date formatting.
`return_fields_by_id`	If `TRUE`, use field IDs as keys instead of names.
`token`	Personal access token (resolved via `air_token()` if `NULL`).
`progress`	Logical or `NULL`. If `TRUE`, shows a cli progress bar for batch operations. If `NULL` (default), uses option `airtable2.progress.bar` or env var `AIRTABLE2_PROGRESS_BAR` (both default to `FALSE`).

Value

A list of record objects (each with `id`, `createdTime`, `fields`).

Examples

```
## Not run:
# List all records from a table
records <- at_list_records("appXXXXXXXXXXXXXX", "Contacts")

# Filter records with a formula
records <- at_list_records(
  "appXXXXXXXXXXXXXX", "Contacts",
  formula = "{Active} = TRUE()",
  fields  = c("Name", "Email")
)

## End(Not run)
```

`at_list_views` *List views in a table*

Description

List views in a table

Usage

```
at_list_views(base_id, table_id, token = NULL)
```

Arguments

`base_id`	Base ID.
`table_id`	Table ID.
`token`	Personal access token (resolved via `air_token()` if `NULL`).

Value

A tibble with columns `id`, `name`, and `type`.

Examples

```
## Not run:
views <- at_list_views("appXXXXXXXXXXXXXX", "tblXXXXXXXXXXXXXX")
views$name

## End(Not run)
```

`at_sitrep` *Summarize Airtable situation report*

Description

Safely probes the current token to determine what workspaces and bases are accessible. Modelled after `usethis::git_sitrep()` and `usethis::proj_sitrep()`. Useful for debugging authentication issues and understanding the scope of the current token.

Usage

```
at_sitrep(token = NULL)
```

Arguments

`token`	Personal access token (resolved via `air_token()` if `NULL`).

Value

A list with:

- `user`: User info from `at_whoami()` (id, and `email`/`scopes` if the token grants them)
- `scopes`: Character vector of token scopes, or `NULL` if not exposed
- `bases`: Tibble of accessible bases (`id`, `name`, `permissionLevel`)
- `error`: `NULL` if successful, otherwise a message string

Note: the non-enterprise Airtable API does not expose workspace names or the workspace each base belongs to, so workspaces cannot be enumerated from a token alone.

Examples

```
## Not run:
# Check what your current token can access
info <- at_sitrep()
print(info$user)
print(info$scopes)
head(info$bases)

## End(Not run)
```

`at_update_field` *Update field metadata*

Description

Update field metadata

Usage

```
at_update_field(
  base_id,
  table_id,
  field_id,
  name = NULL,
  description = NULL,
  token = NULL
)
```

Arguments

`base_id`	Base ID.
`table_id`	Table ID.
`field_id`	Field ID.
`name`	New field name (or `NULL` to leave unchanged).
`description`	New description (or `NULL` to leave unchanged).
`token`	Personal access token (resolved via `air_token()` if `NULL`).

Value

The updated field object.

Examples

```
## Not run:
at_update_field(
  base_id     = "appXXXXXXXXXXXXXX",
  table_id    = "tblXXXXXXXXXXXXX",
  field_id    = "fldXXXXXXXXXXXXX",
  name        = "Full Name",
  description = "First and last name"
)

## End(Not run)
```

`at_update_records` *Update multiple records*

Description

Updates records using PATCH (partial) or PUT (destructive). Handles auto-batching in groups of 10. Also supports upsert via the `upsert_fields` argument.

Usage

```
at_update_records(
  base_id,
  table_id,
  records,
  method = c("PATCH", "PUT"),
  typecast = FALSE,
  upsert_fields = NULL,
  token = NULL,
  progress = NULL
)
```

Arguments

`base_id`	Base ID (e.g., `"appXXXXXX"`).
`table_id`	Table name or ID.
`records`	A list of record objects. Each should be a list with `id` and `fields` elements. For upsert, `id` can be omitted.
`method`	Either `"PATCH"` (partial update) or `"PUT"` (destructive).
`typecast`	If `TRUE`, Airtable will attempt to cast values.
`upsert_fields`	Character vector (1-3 fields) to merge on for upsert. If `NULL`, performs a standard update.
`token`	Personal access token (resolved via `air_token()` if `NULL`).
`progress`	Logical or `NULL`. If `TRUE`, shows a cli progress bar for batch operations. If `NULL` (default), uses option `airtable2.progress.bar` or env var `AIRTABLE2_PROGRESS_BAR` (both default to `FALSE`).

Value

A list with `records`, and (for upserts) `createdRecords` and `updatedRecords` character vectors.

Examples

```
## Not run:
# Patch an existing record
at_update_records(
```

```
  "appXXXXXXXXXXXXXX", "Contacts",
  records = list(list(id = "recXXXXXXXXXXXXXX", fields = list(Age = 31)))
)

# Upsert by Name field
at_update_records(
  "appXXXXXXXXXXXXXX", "Contacts",
  records = list(list(fields = list(Name = "Alice", Age = 31))),
  upsert_fields = "Name"
)

## End(Not run)
```

`at_update_table` *Update table metadata*

Description

Update table metadata

Usage

```
at_update_table(
  base_id,
  table_id,
  name = NULL,
  description = NULL,
  token = NULL
)
```

Arguments

`base_id`	Base ID.
`table_id`	Table ID.
`name`	New table name (or `NULL` to leave unchanged).
`description`	New description (or `NULL` to leave unchanged).
`token`	Personal access token (resolved via `air_token()` if `NULL`).

Value

The updated table object.

Examples

```
## Not run:
at_update_table(
  base_id  = "appXXXXXXXXXXXXXX",
  table_id = "tblXXXXXXXXXXXXXX",
  name     = "Renamed Table"
)

## End(Not run)
```

`at_upload_attachment` *Upload an attachment to a record field*

Description

Upload an attachment to a record field

Usage

```
at_upload_attachment(
  base_id,
  table_id,
  record_id,
  field_id,
  file,
  token = NULL
)
```

Arguments

`base_id`	Base ID.
`table_id`	Table ID or name containing the record.
`record_id`	Record ID.
`field_id`	Field ID or name.
`file`	Path to the file to upload (max 5 MB).
`token`	Personal access token (resolved via `air_token()` if `NULL`).

Value

The attachment object returned by the API.

Examples

```
## Not run:
at_upload_attachment(
  base_id   = "appXXXXXXXXXXXXXX",
  table_id  = "Projects",
  record_id = "recXXXXXXXXXXXXXX",
  field_id  = "Files",
  file      = "report.pdf"
)

## End(Not run)
```

`at_whoami` *Get current user info*

Description

Get current user info

Usage

```
at_whoami(token = NULL)
```

Arguments

`token`	Personal access token (resolved via `air_token()` if `NULL`).

Value

A list with `id` and (if scoped) `email`, `scopes`.

Examples

```
## Not run:
me <- at_whoami()
me$email
me$id

## End(Not run)
```

Index