Package: airtabler (via r-universe)

August 25, 2024

Type Package

Title Interface to the Airtable API

Version 0.2.16

Date 2023-08-17

Author Darko Bergant

Maintainer Collin Schwantes <schwantes@ecohealthalliance.org>

Description Fork from Darko Bergant's package. Provides access to the Airtable (airtable.com) API.

Depends R (>= 3.2.0)

License MIT + file LICENSE

URL https://github.com/ecohealthalliance/airtabler

LazyData TRUE

Encoding UTF-8

Imports glue, dplyr, httr, jsonlite, readxl, curl, purrr, utils, snakecase, tidyselect, rlang, stringr, tibble, deposits, assertthat

Remotes ropenscilabs/deposits

RoxygenNote 7.2.3

Suggests knitr, rmarkdown

VignetteBuilder knitr

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

RemoteUrl https://github.com/ecohealthalliance/airtabler

RemoteRef HEAD

RemoteSha a4028cd5f8a80d73ce07b3de049e0f98777b59cc

Contents

airtabler-package *airtabler: Interface to the Airtable API*

Description

Provides access to the Airtable API (http://airtable.com/api).

Setup

Create and configure the schema of an Airtable base on (http://airtable.com) and check the API on http://airtable.com/api.

API key

Generate the Airtable API token from your Airtable account page (http://airtable.com/create/tokens).

airtabler functions will read the API key from environment variable `AIRTABLE_API_KEY`. To start R session with the initialized environment variable create an `.Renviron` file in your R home with a line like this:

```
AIRTABLE_API_KEY=************
```

To check where your R home is, try `normalizePath("~")`.

The **usethis** and **dotenv** packages are useful for setting environment variables. `usethis::edit_r_environ` allow you to modify the `.Renviron` file. `dotenv::load_dot_env` allows you to load environment variables from a `.env` file. This second approach is especially helpful if you work with multiple tokens.

Usage

Use `airtable` function to get airtable base object or just call primitives `air_get`, `air_insert`, `air_update` and `air_delete` to access your airtable data.

airtable *Get airtable base object*

Description

Creates airtable object with tables and functions

Usage

```
airtable(base, tables)
```

Arguments

`base`	Airtable base
`tables`	Table names in the airtable base (character vector)

Value

Airtable base object with elements named by table names. Each element contains functions

`get`	returns table records, see `air_get` for details
`insert`	insert table record, see `air_insert` for details
`update`	updates table record, see `air_update` for details
`delete`	deletes table record, see `air_delete` for details

Examples

```
## Not run:
TravelBucketList <-
  airtable(
    base = "the_base_id",
    tables = c("Destinations", "Hotels", "Travel Partners")
  )
  hotels <- TravelBucketList$Hotels$get()
  destinations <- TravelBucketList$Destinations$get()

## End(Not run)
```

---

air_create_description_table

Create the descriptive metadata table for the base

Description

Descriptive metadata provides information about the base as a whole, who created it, why, when, where can data be accessed, keywords, what license governs data use, etc. Descriptive metadata facilitates data reuse by providing a point of contact for future users, as well as attributes that allow the data to be entered into searchable catalogs or archives.

Usage

```
air_create_description_table(
  base,
  description,
  table_name = "Description",
  field_descriptions = NA,
  type = "singleLineText",
  options = NA
)
```

Arguments

base	String. Base id
description	Data frame. Description from air_get_base_description* or air_generate_base_description
table_name	String. Name of description table
field_descriptions	Character vector. Descriptions of metadata table fields. If NA, DCMI terms will be used where possible.
type	Character vector. Column types for metadata table fields. see https://airtable.com/developers/web/api/field-model
options	Data frame. Options for fields in metadata table.

Details

DCMI terms can be found here https://www.dublincore.org/specifications/dublin-core/dcmi-terms/

Value

List. Outputs from creating the table and inserting the records

Examples

```
## Not run:
base = "appVjIfAo8AJlfTkx"
table_name= "description"

description <- air_generate_base_description(title = "Example Base",
                                             creator = "Collin Schwantes")

air_create_description_table(base,description,table_name)

## End(Not run)
```

---

air_create_field *Create a new field in a table*

Description

See https://airtable.com/developers/web/api/create-field

Usage

```
air_create_field(
  base,
  table_id,
  name,
  description = NA,
  type = "singleLineText",
  options = NA
)
```

Arguments

base	String. Base id
table_id	String. Table id. Can be found using air_get_schema
name	String. Name of the field
description	String. Description of the field
type	String. Type of field. See https://airtable.com/developers/web/api/field-model
options	Data frame. See https://airtable.com/developers/web/api/field-model

Value

description of newly created field as a list

Examples

```
## Not run:
base_schema <- air_get_schema(base)

base_schema$tables

air_create_field(base,table_id = base_schema$tables$id[[4]],
                 name = "Has Nucleics",
                 description = "Logical. Does this planet have nucleics?",
                 type = "checkbox",
                 options = list(
                   list(
                     "color"= "greenBright",
                     "icon"= "check"
                   )
                 )
)

## End(Not run)
```

`air_create_metadata_table` *Create a new structural metadata table in the base*

Description

Create a new structural metadata table in the base

Usage

```
air_create_metadata_table(
  base,
  meta_data,
  table_name = "Meta Data",
  field_descriptions = NA,
  type = "singleLineText",
  options = NA
)
```

Arguments

`base`	String. Base id
`meta_data`	Data frame. Contains metadata records. From air_generate_metadata*
`table_name`	String. name of the metadata table. default is "Meta Data"
`field_descriptions`	Character vector. Descriptions of metadata table fields
`type`	Character vector. Column types for metadata table fields. see https://airtable.com/developers/web/api/field model
`options`	Data frame. Options for fields in metadata table.

Details

Structural metadata describes the contents of your base and how they are linked. Structural metadata can largely be derived from the base schema.

Value

List with outcome from creating the table and inserting the records

Examples

```
## Not run:
# set base id
base <- "appXXXXXXXX"
# create metadata from api
metadata  <- air_generate_metadata_from_api(base)
# add Meta Data table to base -- will not work if base already has a metadata
```

```
# table
log <- air_create_metadata_table(base,metadata)


## End(Not run)
```

`air_create_table` *A function to create new tables in a base*

Description

Takes a list object with appropriate arguments (see `air_table_template`) converts it to JSON then adds it to the specified base.

Usage

```
air_create_table(base, table_list)
```

Arguments

`base`	String. ID for the base
`table_list`	List. see `air_table_template`

Value

Data frame of table schema

Note

See https://airtable.com/developers/web/api/create-table

Examples

```
## Not run:
base <- "appQ94sELAtFnXPxx"

base_schema <- air_get_schema(base)

tables<- base_schema$tables

field_names <- c("Planet","Chapter","Book", "Known Inhabitants")

field_desc <- c("Name of planet in Foundation Series",
                "Chapters where planet is referenced",
                "Books where planet is referenced",
                "Characters mentioned as living on or being from that planet")
```

```
field_types <- c("singleLineText",rep("multipleRecordLinks",3))

field_options <- c(NA,list(
  list(
    linkedTableId = tables[tables$name == "Chapter","id"]
  )
),
list(
  list(
    linkedTableId = tables[tables$name == "Book","id"]
  )
),
list(
  list(
    linkedTableId = tables[tables$name == "Character","id"]
  )
)
)

field_df<- air_fields_df_template(name = field_names,
                                  description = field_desc,
                                  type = field_types,
                                  options = field_options)

table_list <- air_table_template(table_name = "Planet",
                                 description = "Planets of Foundation",
                                 fields_df = field_df)

air_create_table(base, table_list)


## End(Not run)
```

`air_delete` *Delete a record*

Description

Deletes a record and returns the deleted record id if the call succeeded.

Usage

```
air_delete(base, table_name, record_id)
```

Arguments

`base`	Airtable base
`table_name`	Table name
`record_id`	Id of the record to be deleted

`air_download_attachments` *Download Airtable file attachments*

Description

Download an attachment stored in air tables. Returns original dataframe with an additional field called attachment_file_paths. The attachment_file_paths field is of class list so it can handle multiple attachments per record. File paths are prepended with record ids so that all file names are unique.

Usage

```
air_download_attachments(
  x,
  field,
  dir_name = "downloads",
  include_attachment_id = TRUE,
  ...
)
```

Arguments

`x`	Data frame. Output from air_get or fetch_all.
`field`	String. Name of field with file attachments in base
`dir_name`	String. Where should files be downloaded to? Will create the folder if it does not exist. Folders created are recursively.
`include_attachment_id`	Logical. Should you include the airtable attachment ID to guarantee all file names are unique? Default is true.
`...`	reserved for additional arguments.

Value

Returns x with an additional field called attachment_file_paths

Examples

```
## Not run:

base <- "appXXXXXXXXX"
table_name <- "Table With Attachments"

table_original  <- air_get(base,table_name)

table_with_file_paths <- air_download_attachments(x = table_with_attachments,
                        field = "attachment_field",
```

```
                        dir_name = "downloads")

table_with_file_paths$attachment_file_paths


## End(Not run)
```

`air_dump` *Dump all tables from a base into R*

Description

Dump all tables from a base into R

Usage

```
air_dump(
  base,
  metadata = NULL,
  description = NULL,
  add_missing_fields = TRUE,
  download_attachments = TRUE,
  attachment_fields = NULL,
  polite_downloads = TRUE,
  field_names_to_snakecase = TRUE,
  ...
)
```

Arguments

`base`	String. ID for your base from Airtable. Generally 'appXXXXXXXXXXXXXX'
`metadata`	Data.frame.Data frame with structural metadata - describes relationship between tables and fields. Can be left as NULL if base already contains a table called meta data.
`description`	Data.frame. Data frame with descriptive metadata - describes whats in your base and who made it. Can be left as NULL if base already contains a table called description.
`add_missing_fields`	Logical. Should fields described in the metadata data.frame be added to corresponding tables?
`download_attachments`	Logical. Should attached files be downloaded?
`attachment_fields`	Optional. character vector. What field(s) should files be downloaded from? Default is to download all fields with type multipleAttachments in metadata.

`polite_downloads`
: Logical. Use if downloading many files. Sets a delay so that server is not overwhelmed by requests.

`field_names_to_snakecase`
: Logical. Should field names be converted to snake case?

`...`
: Additional arguments to pass to air_download_attachments

Value

List of data.frames. All tables from metadata plus the description and metadata tables.

Note

To facilitate joining on ids, see purrr::as_vector for converting list type columns to vectors and tidyr::unnest for expanding list columns.

---

air_dump_to_csv *Save air_dump output to csv*

Description

Saves data.frames from air_dump to csv files. File names are determined by the names of the list objects from air_dump. Files will be saved in folder with a unique name, inside the folder specified by `output_dir`. The unique name is generated from a hash of the air_dump output.

Usage

```
air_dump_to_csv(
  table_list,
  output_dir = "outputs",
  attachments_dir = NULL,
  overwrite = FALSE,
  output_id = NULL,
  names_to_snake_case = TRUE
)
```

Arguments

`table_list`
: List. List of data.frames output from `air_dump`

`output_dir`
: String. Folder containing output files

`attachments_dir`
: String. What folder are base attachments stored in?

`overwrite`
: Logical. Should outputs be overwritten if they already exist?

`output_id`
: String. Optional identifier for the data set - if NULL an ID will be generated using a hash of the data.

`names_to_snake_case`
: Logical. Should field and table names be converted to snake_case?

Value

Vector of file paths

`air_dump_to_json` *Dump all tables from a base into json files*

Description

Essentially air_get without converting to Rs. Does not add fields with empty values.

Usage

```
air_dump_to_json(
  base,
  metadata,
  description = NULL,
  output_dir = "outputs",
  overwrite = FALSE
)
```

Arguments

`base`	String. ID for your base from Airtable. Generally 'appXXXXXXXXXXXXX'
`metadata`	Data.frame.Data frame with structural metadata - describes relationship between tables and fields.
`description`	Data.frame. Data frame with descriptive metadata - describes whats in your base and who made it. Can be left as NULL if base already contains a table called description
`output_dir`	String. Where should json files be saved?
`overwrite`	Logical. If data are not unique, should files be overwritten?

Value

List of data.frames. All tables from metadata plus the description and metadata tables.

`air_expand_csv_arrays` *Expand arrays stored in CSVs*

Description

This function helps users work with airtable data that has been exported to CSVs. Because airtable uses nested data structures (json arrays), the data must be flattened to be stored in a csv. The standard way to store arrays in a csv is to wrap the array in quotes and separate each item with commas. So an array stored in a csv would look like "item 1,item 2,...,item n". This function will convert arrays stored in csvs to either a list or a vector and removes the surrounding quotes.

Usage

```
air_expand_csv_arrays(x, simplify_to_vector = FALSE)
```

Arguments

`x`	Character. likely a vector or field in a dataframe.
`simplify_to_vector`	Logical. Should expanded arrays be converted from lists to vectors? For lists with multiple elements at a given position, the length of the output may be greater than the length of the input. See [tidyr::unnest()] for expanding list columns.

Value

A vector or list of expanded arrays.

Examples

```
# example vector data
x <- c("item 1,item 2,item 3","apple,orange,banana","1,2,3","")

# to list
air_expand_csv_arrays(x)

# to vector
air_expand_csv_arrays(x,simplify_to_vector =  TRUE)
```

air_fields_df_template *Template for for creating a table from a tibble*

Description

Convenience function for creating the content of tables that will created or updated viaAPI.

Usage

```
air_fields_df_template(name, description, type, options = NA)
```

Arguments

name	String. Names of fields in the table
description	String. Descriptions of fields
type	String. Type of columns. For values see https://airtable.com/developers/web/api/model/field-type
options	List. Options will be converted from lists to JSON. For field options see https://airtable.com/developers/web/api/field-model

Value

Tibble with attributes required for fields in a table

Examples

```
## Not run:
base <- "appQ94sELAtFnXPxx"

base_schema <- air_get_schema(base)

tables<- base_schema$tables

field_names <- c("Planet","Chapter","Book", "Known Inhabitants")

field_desc <- c("Name of planet in Foundation Series",
                "Chapters where planet is referenced",
                "Books where planet is referenced",
                "Characters mentioned as living on or being from that planet")

field_types <- c("singleLineText",rep("multipleRecordLinks",3))

field_options <- c(NA,list(
  list(
    linkedTableId = tables[tables$name == "Chapter","id"]
  )
),
```

```
  list(
    list(
      linkedTableId = tables[tables$name == "Book","id"]
    )
  ),
  list(
    list(
      linkedTableId = tables[tables$name == "Character","id"]
    )
  )
  )

  field_df<- air_fields_df_template(name = field_names,
                                    description = field_desc,
                                    type = field_types,
                                    options = field_options)

  table_list <- air_table_template(table_name = "Planet",
                                    description = "Planets of Foundation",
                                    fields_df = field_tables)

  air_create_table(base, table_list)

  ## End(Not run)
```

`air_fields_list_from_template` *Convert field data frame to list*

Description

Converts the field data frame to a list of easier translation to JSON

Usage

```
air_fields_list_from_template(df)
```

Arguments

`df`	Data frame. From air_fields_df_template

Value

List. Structured for easy parsing into JSON

Examples

```
## Not run:
base <- "appQ94sELAtFnXPxx"

base_schema <- air_get_schema(base)

tables<- base_schema$tables

field_names <- c("Planet","Chapter","Book", "Known Inhabitants")

field_desc <- c("Name of planet in Foundation Series",
                "Chapters where planet is referenced",
                "Books where planet is referenced",
                "Characters mentioned as living on or being from that planet")

field_types <- c("singleLineText",rep("multipleRecordLinks",3))

field_options <- c(NA,list(
  list(
    linkedTableId = tables[tables$name == "Chapter","id"]
  )
),
list(
  list(
    linkedTableId = tables[tables$name == "Book","id"]
  )
),
list(
  list(
    linkedTableId = tables[tables$name == "Character","id"]
  )
)
)

field_df <- air_fields_df_template(name = field_names,
                                   description = field_desc,
                                   type = field_types,
                                   options = field_options)

fields_list <- air_fields_list_from_template(df = fields_df)


## End(Not run)
```

`air_generate_base_description`

Generate descriptive metadata

Description

Creates a data.frame that describes the base. Descriptive metadata provides information about the base as a whole: who created it, why, when, where can data be accessed, keywords, what license governs data use, etc. Descriptive metadata facilitates data reuse by providing a point of contact for future users, as well as attributes that allow the data to be entered into searchable catalogs or archives.

Usage

```
air_generate_base_description(
  title = NA,
  creator = NA,
  created = NA,
  primary_contact = NA,
  email = NA,
  description = NA,
  contributor = NA,
  identifier = NA,
  license = NA,
  ...
)
```

Arguments

`title`	String. Title is a property that refers to the name or names by which a resource is formally known.
`creator`	String. Person or people who created the base
`created`	String. When was the base created?
`primary_contact`	String. Person or entity primarily responsible for making the content of a resource
`email`	String. Email of primary_contact
`description`	String. This property refers to the description of the content of a resource. The description is a potentially rich source of indexable terms and assist the users in their selection of an appropriate resource.
`contributor`	String. An entity responsible for making contributions to the resource.
`identifier`	String. An unambiguous reference to the resource within a given context.
`license`	String. A legal document giving official permission to do something with the resource. "CC BY 4.0"
`...`	String. Additional descriptive metadata elements. See details. Additional elements can be added as name pair values e.g. `isPartOf = "https://doi.org/00.00000/MyPaper01"`, `isReferencedBy = "https://doi.org/10.48321/MyDMP01"`

Details

See dublin core for inspiration about additional attributes.

Value

data.frame with descriptive metadata

Examples

```
air_generate_base_description(title = "My Awesome Base" ,
 primary_contact= "Base Creator/Maintainer",
 email = "email@example.com",
 base_description = "This base contains my awesome data
 from a project studying XXX in YYY. Data in the base were collected
 from 1900-01-01 to 1990-01-01 by researchers at Some Long Term Project.",
 is_part_of = "https://doi.org/10.48321/MyDMP01",
 isReferencedBy = "https://doi.org/10.5072/zenodo_sandbox.1062705"
 )
```

---

air_generate_metadata_from_api

Generate structural metadata from the api

Description

Structural metadata describes the contents of your base and how they are linked. The structural metadata are created from the base schema. The nested schema structure is flattened into a more user-friendly table which can then be inserted as a table into the base with `air_created_metadata_table` and/or used in a data export with `air_dump`.

Usage

```
air_generate_metadata_from_api(
  base,
  metadata_table_name = "Meta Data",
  include_metadata_table = FALSE,
  field_names_to_snake_case = TRUE
)
```

Arguments

`base`	String. Base id
`metadata_table_name`	String. Name of exisiting structural metadata table if it exists
`include_metadata_table`	Logical. Should the structural metadata table be included in the metadata?
`field_names_to_snake_case`	Logical. Should the field names in the metadata table be snake_case?

Details

This function requires that the api token has the ability to read the base schema.

Value

A data frame with metadata

Examples

```
## Not run:

base <- "appXXXXXXXX"
metadata  <- air_generate_metadata_from_api(base)


## End(Not run)
```

`air_generate_metadata_from_tables`
Generated Metadata from table names

Description

Deprecated: Use `air_generate_metadata_from_api`

Usage

```
air_generate_metadata_from_tables(base, table_names, limit = 1)
```

Arguments

`base`	String. ID for your base from Airtable. Generally 'appXXXXXXXXXXXXXX'
`table_names`	Vector of strings. The names of your tables. eg c("table 1","table 2", etc.)
`limit`	Number from 1-100. How many rows should we pull from each table to create the metdata? Keep in mind that the airtable api will not return fields with "empty" values - "", false, or []. Code runs faster if fewer rows are pulled.

Details

Generates a structural metadata table - the metadata that describes how tables and fields fit together. Does not include field types.

For information about creating metadata tables in your base see the EHA MA Handbook

Value

data.frame with structural metadata.

`air_get` *Get a list of records or retrieve a single*

Description

Retrieve records or a single record from a table. If you provide a record_id, you cannot specify fields, views, or filterFormulas.

Usage

```
air_get(
  base,
  table_name,
  record_id = NULL,
  limit = NULL,
  offset = NULL,
  view = NULL,
  fields = NULL,
  sortField = NULL,
  sortDirection = NULL,
  filterByFormula = NULL,
  combined_result = TRUE
)
```

Arguments

`base`	Airtable base
`table_name`	Table name
`record_id`	(optional) Use record ID argument to retrieve an existing record details. See https://airtable.com/developers/web/api/get-record
`limit`	(optional) A limit on the number of records to be returned. Limit can range between 1 and 100.
`offset`	(optional) Page offset returned by the previous list-records call. Note that this is represented by a record ID, not a numerical offset.
`view`	(optional) The name or ID of the view
`fields`	List. (optional) Only data for fields whose names are in this list will be included in the records. Does not work when retrieving individual records with `record_id`
`sortField`	(optional) The field name to use for sorting
`sortDirection`	(optional) "asc" or "desc". The sort order in which the records will be returned. Defaults to asc.
`filterByFormula`	String. Use a formula to filter results. See https://support.airtable.com/hc/en-us/articles/223247187-How-to-sort-filter-or-retrieve-ordered-records-in-the-API this parameter to reduce the amount of data transferred.

`combined_result`
: If TRUE (default) all data is returned in the same data. If FALSE table fields are returned in separate `fields` element.

Details

You can retrieve records in an order of a view by providing the name or ID of the view in the view query parameter. The results will include only records visible in the order they are displayed.

Value

A data frame with records or a list with record details if `record_id` is specified.

---

air_get_attachments *Get Airtable file attachments*

Description

Extract the contents of an attachment stored in Airtable. Currently only setup to work with Excel files. Planned expansion to other file types. For excel files, returns a named list.

Usage

```
air_get_attachments(
  base,
  table_name,
  field,
  download_file = FALSE,
  include_attachment_id = TRUE,
  dir_name = "downloads",
  extract_type = "excel",
  extract_field = "excel_extract",
  skip = 0,
  parse_all_sheets = FALSE,
  ...
)
```

Arguments

`base`	String. ID for the base or app to be fetched
`table_name`	String. Name of the table to be fetched from the base
`field`	String. Name of field with file attachments in base
`download_file`	Logical. Should files be downloaded?
`include_attachment_id`	Logical. Should the attachment ID be included in the file name? Default is true to ensure unique file names.

dir_name	String. Where should files be downloaded to? Will create the folder if it does not exist.
extract_type	String. File type to be extracted. Should be one of: excel
extract_field	String. Name of extract field that will be created
skip	Numeric. How many lines should be skipped? See readxl::read_excel skip.
parse_all_sheets	Logical. Should all sheets in spreadsheet be parsed?
...	Additional arguments to pass to air_get

Value

named list of data frames

See Also

air_download_attachments

Examples

```
## Not run:

base <- "appXXXXXXXXX"
table_name <- "table with excel attachments"

 table_with_attachments <- air_get_attachments(base,table_name, field = "attachment_field" )


## End(Not run)
```

---

air_get_base_description_from_table

Get base description from table

Description

Pull a table that has descriptive metadata. Requires the following fields: "title","primary_contact","email","description"

Usage

```
air_get_base_description_from_table(
  base,
  table_name,
  field_names_to_snakecase = TRUE
)
```

Arguments

base	String. ID for your base from Airtable. Generally 'appXXXXXXXXXXXXXX'
table_name	String. Name of descriptive metadata table - the metadata that describes the base and provides attribution
field_names_to_snakecase	Logical. Should field names be converted to snakecase?

Value

data.frame with descriptive metadata.

Examples

```
## Not run:
base <- "appXXXXXXXX"
table_name <- "Description"
air_get_base_description_from_table(base, table_name)

## End(Not run)
```

---

air_get_id_from_url *Get an ID from a URL*

Description

General function for parsing airtable URLs to find base, table, view, or record id's

Usage

```
air_get_id_from_url(url, pattern, id_type, split_pattern = "/|\\?")

air_get_base_id_from_url(url, pattern = "^app\\w{13}")

air_get_table_id_from_url(url, pattern = "^tbl\\w{13}")

air_get_view_id_from_url(url, pattern = "^viw\\w{13}")

air_get_record_id_from_url(url, pattern = "^rec\\w{13}")
```

Arguments

url	String. A url generated by airtable
pattern	String. A regex pattern for identifying the type of id you would like to get
id_type	String. One of base_id, table_id, view_id, or record_id.
split_pattern	String. Where should the URL be split? default is forward slashes "/" and questionmarks "?"

Value

String

Functions

- `air_get_base_id_from_url()`: Get the base id
- `air_get_table_id_from_url()`: Get the table id
- `air_get_view_id_from_url()`: Get the view id
- `air_get_record_id_from_url()`: Get the record id

Examples

```
url <- "https://airtable.com/apphDEokVZ9gvPNFk/tblaKC1ADBafoHVXN/viwteUgD7vaMBruHR/recMzdoM43RVRWybD?blocks=hid

# General function for parsing url components
air_get_id_from_url(url, '^app',id_type = "base_id")
# Get different components
air_get_base_id_from_url(url)
air_get_table_id_from_url(url)
air_get_view_id_from_url(url)
air_get_record_id_from_url(url)
```

---

air_get_json *Get a list of records or retrieve a single record as JSON*

Description

Returns JSON objects from GET requests

Usage

```
air_get_json(
  base,
  table_name,
  record_id = NULL,
  limit = NULL,
  offset = NULL,
  view = NULL,
  fields = NULL,
  sortField = NULL,
  sortDirection = NULL,
  combined_result = TRUE,
  pretty = FALSE
)
```

Arguments

base	Airtable base
table_name	Table name
record_id	(optional) Use record ID argument to retrieve an existing record details
limit	(optional) A limit on the number of records to be returned. Limit can range between 1 and 100.
offset	(optional) Page offset returned by the previous list-records call. Note that this is represented by a record ID, not a numerical offset.
view	(optional) The name or ID of the view
fields	(optional) Only data for fields whose names are in this list will be included in the records. If you don't need every field, you can use
sortField	(optional) The field name to use for sorting
sortDirection	(optional) "asc" or "desc". The sort order in which the records will be returned. Defaults to asc.
combined_result	If TRUE (default) all data is returned in the same data. If FALSE table fields are returned in separate `fields` element.
pretty	Logical. Should JSON be returned in human readable form? this parameter to reduce the amount of data transferred.

Value

A data frame with records or a list with record details if `record_id` is specified.

---

air_get_metadata_from_table

Pull the metadata table from Airtable

Description

Airtable allows all users to access the metadata API. The recommended workflow for creating this table is to use air_generate_metadata_from_api to extract the structural metadata from the base schema and then use air_create_metadata_table to add the table to your base.

Usage

```
air_get_metadata_from_table(
  base,
  table_name,
  add_id_field = FALSE,
  field_names_to_snakecase = TRUE
)
```

Arguments

base	String. ID for your base from Airtable. Generally 'appXXXXXXXXXXXXX'
table_name	String. Name of structural metadata table - the metadata that describes how tables and fields fit together.
add_id_field	Logical. If true, an "id" field is added to each table
field_names_to_snakecase	Logical. If true, values in the field_names column and the field in the metadata table themselves are are converted to snake_case

Details

For information about creating metadata tables in your base see the EHA MA Handbook

Requires the following fields: table_name, field_name

Value

data.frame with metadata table

---

air_get_schema *Get base schema*

Description

Get the schema for the tables in a base. This is a wrapper for the api call Get base schema.

Usage

```
air_get_schema(base, ...)
```

Arguments

base	String. Airtable base ID
...	reserved for additional parameters

Value

list of schema

Using Metadata API

Metadata api is currently available to all users.

air_insert *Insert a new record*

Description

Creates a new record and returns the created record object if the call succeeded, including a record ID which will uniquely identify the record within the table.

Usage

```
air_insert(base, table_name, record_data)

air_insert_data_frame(base, table_name, records, typecast)

multiple(x)
```

Arguments

base	String. Airtable base
table_name	String. Table name
record_data	Named list of values. You can include all, some, or none of the field values
records	Dataframe. Contains records you would like to insert
typecast	Logical. Should airtable make new values for select type fields?
x	Object to be marked as a multiple value field

air_list_bases *Get list of bases for an Token*

Description

Each token you provisision is given access to a certain set of bases or workspaces. This function lists all bases associated with a token.

Usage

```
air_list_bases(request_url = "https://api.airtable.com/v0/meta/bases")
```

Arguments

request_url	String. URL for api endpoint

Value

list. List of bases a token can access.

Examples

```
## Not run:
air_list_bases()

## End(Not run)
```

`air_make_json` *Make JSON for API*

Description

Make JSON that is compatible with the Airtable API.

Usage

```
air_make_json(
  base,
  table_name,
  record_data,
  record_id = NULL,
  method = "POST",
  typecast = TRUE
)
```

Arguments

`base`	String. Base in airtable
`table_name`	String. Table in airtable
`record_data`	Dataframe, list, or vector. Data to be converted to JSON
`record_id`	String or vector of strings. Records to be manipulated
`method`	String. "PATCH" is necessary for `air_update`
`typecast`	Logical. Should the typecast option be TRUE or FALSE? Typecast allows you to add new options to select type fields.

Value

JSON with record data

air_make_request *Make an HTTP request*

Description

Properly encodes HTTP requests

Usage

```
air_make_request(
  base,
  table_name,
  json_record_data,
  record_id = NULL,
  method = c("POST", "PATCH", "DELETE")
)
```

Arguments

base	String. Base in airtable
table_name	String. Table in airtable
json_record_data	json or string. JSON formatted text with record data
record_id	String or vector of strings. Record id
method	String. One of "POST", "PATCH", or "DELETE"

Value

Status of HTTP request

air_post *Get a list of records*

Description

Retrieve records where the request url would be over 16k characters (e.g. complicated formula) or has more than 21 fields listed in the request.

Usage

```
air_post(
  base,
  table_name,
  limit = NULL,
  offset = NULL,
  view = NULL,
  fields = NULL,
  sortField = NULL,
  sortDirection = NULL,
  filterByFormula = NULL,
  combined_result = TRUE
)
```

Arguments

`base`	String. Airtable base
`table_name`	String. Table name
`limit`	Numeric. (optional) A limit on the number of records to be returned. Limit can range between 1 and 100.
`offset`	Numeric. (optional) Page offset returned by the previous list-records call. Note that this is represented by a record ID, not a numerical offset.
`view`	String. (optional) The name or ID of the view
`fields`	List. (optional) Only data for fields whose names are in this list will be included in the records. Does not work when retrieving individual records with `record_id`
`sortField`	String. (optional) The field name to use for sorting
`sortDirection`	String. (optional) "asc" or "desc". The sort order in which the records will be returned. Defaults to asc.
`filterByFormula`	String. Use a formula to filter results. See https://support.airtable.com/hc/en-us/articles/223247187-How-to-sort-filter-or-retrieve-ordered-records-in-the-API this parameter to reduce the amount of data transferred.
`combined_result`	Logical. If TRUE (default) all data is returned in the same data. If FALSE table fields are returned in separate `fields` element.

Details

You can retrieve records in an order of a view by providing the name or ID of the view in the view query parameter. The results will include only records visible in the order they are displayed.

Value

A data frame with records

See Also

[air_get()]

`air_select` *Select*

Description

Select records from table

Usage

```
air_select(
  base,
  table_name,
  record_id = NULL,
  fields = NULL,
  filterByFormula = NULL,
  maxRecord = NULL,
  sort = NULL,
  view = NULL,
  pageSize = NULL,
  offset = NULL,
  combined_result = TRUE
)
```

Arguments

`base`	Airtable base
`table_name`	Table name
`record_id`	(optional) Use record ID argument to retrieve an existing record details
`fields`	(optional) Only data for fields whose names are in this list will be included in the records. If you don't need every field, you can use this parameter to reduce the amount of data transferred.
`filterByFormula`	(optional) A formula used to filter records.
`maxRecord`	(optional) The maximum total number of records that will be returned.
`sort`	A list of sort objects that specifies how the records will be ordered.
`view`	(optional) The name or ID of the view defined in the table
`pageSize`	(optional) The number of records returned in each request. Must be less than or equal to 100. Default is 100.
`offset`	(optional) To fetch the next page of records set this argument with a value of offset element from previous response
`combined_result`	If TRUE (default) all data is returned in the same data frame. If FALSE table fields are returned in separate `fields` element.

Value

A data frame with records or a list with record details if `record_id` is specified.

View

You can retrieve records in an order of a view by providing the name or ID of the view in the view query parameter. The results will include only records visible in the order they are displayed.

Filter by formula

The formula will be evaluated for each record, and if the result is not 0, false, "", NaN, [], or #Error! the record will be included in the response. If combined with view, only records in that view which satisfy the formula will be returned. For example, to only include records where Country isn't empty, pass in: NOT(Country = '')

Sorting

Each sort object must have a field key specifying the name of the field to sort on, and an optional direction key that is either "asc" or "desc". The default direction is "asc". For example, to sort records by Country, pass in: `list(field = "Country", direction = "desc")`

`air_table_template` *Template for lists that describe tables in Airtable*

Description

Template for lists that describe tables in Airtable

Usage

```
air_table_template(table_name, description, fields_df)
```

Arguments

`table_name`	String. Name of table
`description`	String. Description of the table
`fields_df`	Data frame. Data frame describing the field in a table. Should contain a name, description,type, and options field. if

Value

List with table name, description, and fields

Examples

```
## Not run:
base <- "appQ94sELAtFnXPxx"

base_schema <- air_get_schema(base)

tables<- base_schema$tables

field_names <- c("Planet","Chapter","Book", "Known Inhabitants")

field_desc <- c("Name of planet in Foundation Series",
                "Chapters where planet is referenced",
                "Books where planet is referenced",
                "Characters mentioned as living on or being from that planet")

field_types <- c("singleLineText",rep("multipleRecordLinks",3))

field_options <- c(NA,list(
  list(
    linkedTableId = tables[tables$name == "Chapter","id"]
  )
),
list(
  list(
    linkedTableId = tables[tables$name == "Book","id"]
  )
),
list(
  list(
    linkedTableId = tables[tables$name == "Character","id"]
  )
)
)

field_df<- air_fields_df_template(name = field_names,
                                  description = field_desc,
                                  type = field_types,
                                  options = field_options)

table_list <- air_table_template(table_name = "Planet",
                                 description = "Planets of Foundation",
                                 fields_df = field_tables)

air_create_table(base, table_list)

## End(Not run)
```

`air_update` *Update a record*

Description

Updates a new record. Any fields that are not included will not be updated.

Usage

```
air_update(base, table_name, record_id, record_data)
```

Arguments

base	Airtable base
table_name	Table name
record_id	An id of the record
record_data	Named list of values. You can include all, some, or none of the field values

---

air_update_data_frame *Update records from a dataframe*

Description

Updates the values in a table by overwriting their current contents.

Usage

```
air_update_data_frame(base, table_name, record_ids, records)
```

Arguments

base	String. Airtable base
table_name	String. Table name
record_ids	Vector of strings. Records to be modified
records	Dataframe. Values to update

Value

Status of HTTP request

air_update_description_table *Update the description table*

Description

Update the descriptive metadata table in airtable

Usage

```
air_update_description_table(
  base,
  description,
  table_name = "Description",
  join_field = "title",
  record_id_field = "id"
)
```

Arguments

base	String. Base id
description	Data frame. Contains updated description
table_name	String. Name of description table
join_field	String. Field to perform join on
record_id_field	String. Name of the record id field

Value

list that logs updates

Examples

```
## Not run:

base <- "appXXXXXXXX"
table_name <- "Description"
# get description from table
description  <- air_get_base_description_from_table(base, table_name)
# update the identifier field
description$identifier <- "fake.doi.xyz/029940"
# update the table
air_update_description_table(base,description)



## End(Not run)
```

`air_update_field` *Update a field name and/or description*

Description

Must update either the name or the description. See "https://airtable.com/developers/web/api/update-field" for more details.

Usage

```
air_update_field(base, table_id, field_id, name = NULL, description = NULL)
```

Arguments

`base`	String. Base id
`table_id`	String. ID for table that contains the field to be updated
`field_id`	String. ID of field to be updated
`name`	String. updated name (optional)
`description`	String. updated description (option)

Value

List. Describes the changes that happened to the field

Examples

```
## Not run:
base <- "appVjIfAo8AJlfTkx"

schema <- air_get_schema("appVjIfAo8AJlfTkx")

table_id <- schema$tables[1,c("id")]

field_id <- schema$tables$fields[[1]][2,]$id

## update name and description

name <- "New Name"

description <- "Updated Description"

out <- air_update_field(base = base,table_id = table_id,field_id = field_id,
name = name, description = description)

### just name
```

```
name <- "New New Name"

out <- air_update_field(base = base,table_id = table_id,field_id = field_id,
name = name)


## just description

description <- "Better description"

out <- air_update_field(base = base,table_id = table_id,field_id = field_id,
description = description)

## set name to number

name <- 1234

out <- air_update_field(base = base,table_id = table_id,field_id = field_id,
name = name)


# set description to number

description <- 1234

out <- air_update_field(base = base,table_id = table_id,field_id = field_id,
description = description)

# bulk update names and descriptions from a data frame

field_ids <- schema$tables$fields[[1]]$id

field_names <- sprintf("%s_bulk_update",schema$tables$fields[[1]]$name)

field_descriptions <- sprintf("%s BULK UPDATE",
schema$tables$fields[[1]]$description)

df <- data.frame("field_id"= field_ids,"name"=field_names,
"description"=field_descriptions)

purrr::pmap(df,function(field_id,name,description){
  air_update_field(base = base,table_id = table_id,field_id = field_id,
  name = name, description = description)
})

## End(Not run)
```

---

air_update_metadata_table

Update the structural metadata table

Description

Update the structural metadata table

Usage

```
air_update_metadata_table(
  base,
  meta_data,
  table_name = "Meta Data",
  join_field = "field_id",
  record_id_field = "id"
)
```

Arguments

base	String. Base id
meta_data	Data frame. Contains metadata records. From air_generate_metadata*
table_name	String. Name of metadata table
join_field	String. Name of field to join new and current metadata. Likely field_id
record_id_field	String. Name of record id field. Like id

Value

List. Log of results for updating metadata

Examples

```
## Not run:
base = "appVjIfAo8AJlfTkx"
metadata <- air_generate_metadata_from_api(base = base)
air_update_metadata_table(base,metadata)

## End(Not run)
```

---

fetch_all *Fetch All Records in an Airtable*

Description

Airtable limits the number of records that can be pulled from a base to 100. This function pulls records based on a query then checks if there is an offset value. While there is an offset value, it uses that value to generate the next query, thus moving down the records until all records have been fetched from the database.

Usage

```
fetch_all(base, table_name, ...)
```

Arguments

base	String. ID for the base or app to be fetched
table_name	String. Name of the table to be fetched from the base
...	Additional arguments to pass to [air_get()]. view is a commonly used additional argument.

Value

dataframe

Adding airtable API key to your environment

Airtable requires an api key to fetch data. Generate the airtable API key from your [Airtable account](http://airtable.com/account) page.

airtabler functions will read the API key from environment variable AIRTABLE_API_KEY. To start R session with the initialized environment variable create an .Renviron file in your home directory with a line like this:

AIRTABLE_API_KEY=your_api_key_here

You can use usethis::edit_r_environ() to open and edit your .Renviron file.

Also consider using the 'dotenv' package with a .env file for storing sensitive variables. Remember add to.gitignore or encrypt the .env file to avoid sharing sensitive variables.

Examples

```
# Each base has a fully described API
# app_id <- "appVjIfAo8AJlfTkx" # ID for the base we are fetching.
# Note that you can pass a `view` argument to air_get or fetch_all to get only
# a view of a table (say, only validated records, or some other filtered view),
# e.g.,
# bats <- fetch_all(app_id, "images", view = "Status View")
# talks <- fetch_all(app_id, "images")
```

---

fetch_all_json *Get the full outputs of a table as single json object*

Description

Get the full outputs of a table as single json object

Usage

```
fetch_all_json(base, table_name, ...)
```

Arguments

base	String. Base ID
table_name	String. Table name
...	additional parameters to pass to air_get_json

Value

json as string

Examples

```
## Not run:
base <- "appXXXXXXX"
table_name <- "My Table"

fetch_all_json(base, table_name)


## End(Not run)
```

---

flatten_col_to_chr *Flatten list columns to character*

Description

Similar in spirit to purrr::flatten_chr except that it can handle NULL values in lists and returns outputs that can be written to csv.

Usage

```
flatten_col_to_chr(data_frame)
```

Arguments

data_frame	a data frame, tibble or other data frame like object

Details

Because the outputs are intended for use in CSV files, we must use double quotes to indicate that the commas separating list values do not delimit cells. This conforms to RFC 4180 standard for CSVs. https://datatracker.ietf.org/doc/html/rfc4180

Value

data_frame with list columns converted to character vectors.

Examples

```
data_frame <- data.frame(a = I(list(list("Hello"),
list("Aloha"),
NULL,
list("Hola","Bonjour","Merhaba")
)),
b = 1:4,
c = letters[1:4],
d = I(data.frame(id = 1:4, name = "bob", email = "bob@example.com"))
)

test_df <- flatten_col_to_chr(data_frame)

str(test_df)
```

---

get_offset *Get offset*

Description

Returns airtable offset id from previous select

Usage

```
get_offset(x)
```

Arguments

x	Last result

Value

Airtable offset id

---

get_unique_field_values *Get unique values from a field*

Description

Because the airtable api lacks an easy solution for getting unique values from a field in a table, we have this function which accepts a list of fields and returns their unique values. Currently, if multiple fields are listed, all unique values from all fields will be returned in a single vector. This may change in future iterations.

Usage

```
get_unique_field_values(base, table_name, fields)
```

Arguments

base	String. ID of airtable base
table_name	String. Name of table in base
fields	List. Names of fields

Value

vector of unique values

---

read_excel_url *Read an excel file from URL*

Description

Extends `readxl::read_excel` to allow for reading from a URL.

Usage

```
read_excel_url(url, fileext = ".xslx", parse_all_sheets = FALSE, ...)
```

Arguments

url	String. Url for file
fileext	String. File extension for temp file
parse_all_sheets	Logical. Should all sheets be parsed?
...	additional arguments to pass to `read_excel`

Value

tibble or list of tibbles if parse_all_sheets = TRUE

set_diff *Get items that differ between x and y*

Description

Unlike setdiff, this function creates the union of x and y then removes values that are in the intersect, providing values that are unique to X and values that are unique to Y.

Usage

```
set_diff(x, y)
```

Arguments

x	a set of values.
y	a set of values.

Value

Unique values from X and Y, NULL if no unique values.

Examples

```
a <- 1:3
b <- 2:4

set_diff(a,b)
# returns 1,4

x <- 1:3
y <- 1:3

set_diff(x,y)
# returns NULL
```

Index