# Package 'frictionless'

November 16, 2022

**Title** Read and Write Frictionless Data Packages

**Version** 1.0.2

**Description** Read and write Frictionless Data Packages. A 'Data Package' (<https://specs.frictionlessdata.io/data-package/>) is a simple container format and standard to describe and package a collection of (tabular) data. It is typically used to publish FAIR (<https://www.go-fair.org/fair-principles/>) and open datasets.

**License** MIT + file LICENSE

**URL** https://github.com/frictionlessdata/frictionless-r, https://docs.ropensci.org/frictionless/

**BugReports** https://github.com/frictionlessdata/frictionless-r/issues

**Imports** assertthat, dplyr, glue, httr, jsonlite, purrr, readr (>= 2.1.0), utils, yaml

**Suggests** hms, knitr, lubridate, rmarkdown, testthat (>= 3.0.0)

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.2.1

**NeedsCompilation** no

**Author** Peter Desmet [aut, cre] (<https://orcid.org/0000-0002-8442-8025>), Damiano Oldoni [aut] (<https://orcid.org/0000-0003-3445-7562>), Research Institute for Nature and Forest (INBO) [cph] (https://www.vlaanderen.be/inbo/en-gb/), LifeWatch Belgium [fnd] (https://lifewatch.be), Beatriz Milz [rev] (<https://orcid.org/0000-0002-3064-4486>), João Martins [rev] (<https://orcid.org/0000-0001-7961-4280>)

**Maintainer** Peter Desmet <peter.desmet.work@gmail.com>

**Depends** R (>= 3.5.0)

**Repository** CRAN

**Date/Publication** 2022-11-16 22:20:06 UTC

# R topics documented:



---

`add_resource` *Add a Data Resource*

---

## Description

Adds a Tabular Data Resource to a Data Package. The resource will be a Tabular Data Resource. The resource name can only contain lowercase alphanumeric characters plus `.`, `-` and `_`.

## Usage

```
add_resource(package, resource_name, data, schema = NULL, delim = ",")
```

## Arguments

| | |
|---|---|
| `package` | List describing a Data Package, created with `read_package()` or `create_package()`. |
| `resource_name` | Name of the Data Resource. |
| `data` | Data to attach, either a data frame or path(s) to CSV file(s):<br>• Data frame: attached to the resource as `data` and written to a CSV file when using `write_package()`.<br>• One or more paths to CSV file(s) as a character (vector): added to the resource as `path`. The **last file will be read** with `readr::read_delim()` to create or compare with `schema` and to set `format`, `mediatype` and `encoding`. The other files are ignored, but are expected to have the same structure and properties. |
| `schema` | Either a list, or path or URL to a JSON file describing a Table Schema for the `data`. If not provided, one will be created using `create_schema()`. |
| `delim` | Single character used to separate the fields in the CSV file(s), e.g. `\t` for tab delimited file. Will be set as `delimiter` in the resource CSV dialect, so read functions know how to read the file(s). |

## Value

Provided `package` with one additional resource.

## See Also

Other edit functions: `get_schema()`, `remove_resource()`

## Examples

```
# Load the example Data Package
package <- example_package

# List resources
resources(package)

# Create a data frame
df <- data.frame(
  multimedia_id = c(
    "aed5fa71-3ed4-4284-a6ba-3550d1a4de8d",
    "da81a501-8236-4cbd-aa95-4bc4b10a05df"
  ),
  x = c(718, 748),
  y = c(860, 900)
)

# Add resource "positions" to the Data Package, from the data frame
package <- add_resource(package, "positions", data = df)

# Add resource "positions_2" to the Data Package, with user-defined schema
my_schema <- create_schema(df)
package <- add_resource(package, "positions_2", data = df, schema = my_schema)

# Add resource "observations_2" to the Data Package, from CSV file paths
path_1 <- system.file("extdata", "observations_1.csv", package = "frictionless")
path_2 <- system.file("extdata", "observations_2.csv", package = "frictionless")
package <- add_resource(package, "observations_2", data = c(path_1, path_2))

# List resources ("positions", "positions_2", "observations_2" added)
resources(package)
```

---

# create_package *Create an empty Data Package*

---

## Description

Initiates a list describing a Data Package. This empty Data Package can be extended with metadata and resources (see `add_resource()`). Added resources will make the Data Package meet Tabular Data Package requirements, so `profile` is set to `tabular-data-package`.

## Usage

```
create_package()
```

## Value

List describing a Data Package.

## See Also

Other create functions: `create_schema()`

## Examples

```
# Create a Data Package
package <- create_package()
str(package)
```

---

`create_schema` *Create a Table Schema for a data frame*

---

## Description

Creates a Table Schema for a data frame, listing all column names and types as field names and (converted) types.

## Usage

```
create_schema(data)
```

## Arguments

| | |
|---|---|
| `data` | A data frame. |

## Value

List describing a Table Schema.

## Table schema properties

The Table Schema will be created from the data frame columns:

- `name`: contains the column name.
- `title`: not set.
- `description`: not set.
- `type`: contains the converted column type (see further).
- `format`: not set and can thus be considered `default`. This is also the case for dates, times and datetimes, since `readr::write_csv()` used by `write_package()` will format those to ISO8601 which is considered the default. Datetimes in local or non-UTC timezones will be converted to UTC before writing.
- `constraints`: not set, except for factors (see further).

- `missingValues`: not set. `write_package()` will use the default `""` for missing values.
- `primaryKey`: not set.
- `foreignKeys`: not set.

**Field types:**

The column type will determine the field `type`, as follows:

- `character` as string.
- `Date` as date.
- `difftime` as number.
- `factor` as string with factor levels as `enum`.
- `hms::hms()` as time.
- `integer` as integer.
- `logical` as. boolean.
- `numeric` as number.
- `POSIXct/POSIXlt` as datetime.
- Any other type as any.

## See Also

Other create functions: `create_package()`

## Examples

```
# Create a data frame
df <- data.frame(
  id = c(as.integer(1), as.integer(2)),
  timestamp = c(
    as.POSIXct("2020-03-01 12:00:00", tz = "EET"),
    as.POSIXct("2020-03-01 18:45:00", tz = "EET")
  ),
  life_stage = factor(c("adult", "adult"), levels = c("adult", "juvenile"))
)

# Create a Table Schema from the data frame
schema <- create_schema(df)
str(schema)
```

---

# example_package *Example Data Package*

---

## Description

Example Tabular Data Package with dummy camera trap data organized in 3 Data Resources:

- `deployments`: data stored in `deployments.csv`.
- `observations`: data stored in `observations_1.csv` and `observations_2.csv`, but referenced as URLs.
- `media`: data stored in `data` property.

## Usage

```
example_package
```

## Format

An object of class `list` of length 9.

## Source

https://github.com/frictionlessdata/frictionless-r/tree/main/inst/extdata

## Examples

```
## Not run:
# example_package.rda was created with the code below.
# Note that it must be created using a URL, otherwise all Data Resource paths
# will point to local paths that won't work for other users.
# One can load locally using:
# read_package(
#   system.file("extdata", "datapackage.json", package = "frictionless")
# )
example_package <- read_package(file.path(
  "https://raw.githubusercontent.com/frictionlessdata/frictionless-r",
  "main/inst/extdata/datapackage.json"
))
save(example_package, file = "data/example_package.rda")

## End(Not run)
```

---

# get_schema *Get the Table Schema of a Data Resource*

---

## Description

Returns the Table Schema of a Data Resource (in a Data Package), i.e. the content of its `schema` property, describing the resource's fields, data types, relationships, and missing values. The resource must be a Tabular Data Resource.

## Usage

```
get_schema(package, resource_name)
```

## Arguments

| | |
|---|---|
| `package` | List describing a Data Package, created with `read_package()` or `create_package()`. |
| `resource_name` | Name of the Data Resource. |

## Value

List describing a Table Schema.

## See Also

Other edit functions: `add_resource()`, `remove_resource()`

## Examples

```
# Load the example Data Package
package <- example_package

# Get the Table Schema for the resource "observations"
schema <- get_schema(package, "observations")
str(schema)
```

---

# read_package *Read a Data Package descriptor file (*`datapackage.json`*)*

---

## Description

Reads information from a `datapackage.json` file, i.e. the descriptor file that describes the Data Package metadata and its Data Resources.

## Usage

```
read_package(file = "datapackage.json")
```

## Arguments

| | |
|---|---|
| `file` | Path or URL to a `datapackage.json` file. |

## Value

List describing a Data Package. The function will add a custom property `directory` with the directory the descriptor was read from. It is used as a base path to access resources.

## See Also

Other read functions: `read_resource()`, `resources()`

## Examples

```
# Read a datapackage.json file
package <- read_package(
  system.file("extdata", "datapackage.json", package = "frictionless")
)

# Access the Data Package properties
package$name
package$created

# List resources
resources(package)
```

---

`read_resource` *Read data from a Data Resource into a tibble data frame*

---

## Description

Reads data from a Data Resource (in a Data Package) into a tibble (a Tidyverse data frame). The resource must be a Tabular Data Resource. The function uses `readr::read_delim()` to read CSV files, passing the resource properties `path`, CSV dialect, column names, data types, etc. Column names are taken from the provided Table Schema (`schema`), not from the header in the CSV file(s).

## Usage

```
read_resource(package, resource_name)
```

## Arguments

| | |
|---|---|
| `package` | List describing a Data Package, created with `read_package()` or `create_package()`. |
| `resource_name` | Name of the Data Resource. |

## Value

`dplyr::tibble()` data frame with the Data Resource's tabular data.

## Resource properties

The Data Resource properties are handled as follows:

**Path:**
`path` is required. It can be a local path or URL, which must resolve. Absolute path (`/`) and relative parent path (`../`) are forbidden to avoid security vulnerabilities.
When multiple paths are provided (`"path": [ "myfile1.csv", "myfile2.csv"]`) then data are merged into a single data frame, in the order in which the paths are listed.

**Data:**
If `path` is not present, the function will attempt to read data from the `data` property. `schema` **will be ignored**.

**Name:**
`name` is required. It is used to find the resource with `name = resource_name`.

**Profile:**
`profile` is required to have the value `tabular-data-resource`.

**File encoding:**
`encoding` (e.g. `windows-1252`) is required if the resource file(s) is not encoded as UTF-8. The returned data frame will always be UTF-8.

**CSV Dialect:**
`dialect` properties are required if the resource file(s) deviate from the default CSV settings (see below). It can either be a JSON object or a path or URL referencing a JSON object. Only deviating properties need to be specified, e.g. a tab delimited file without a header row needs:

```
"dialect": {"delimiter": "\t", "header": "false"}
```

These are the CSV dialect properties. Some are ignored by the function:

- `delimiter`: default `,`.
- `lineTerminator`: ignored, line terminator characters `LF` and `CRLF` are interpreted automatically by `readr::read_delim()`, while `CR` (used by Classic Mac OS, final release 2001) is not supported.
- `doubleQuote`: default `true`.
- `quoteChar`: default `"`.
- `escapeChar`: anything but `\` is ignored and it will set `doubleQuote` to `false` as these fields are mutually exclusive. You can thus not escape with `\"` and `""` in the same file.
- `nullSequence`: ignored, use `missingValues`.
- `skipInitialSpace`: default `false`.
- `header`: default `true`.
- `commentChar`: not set by default.
- `caseSensitiveHeader`: ignored, header is not used for column names, see Schema.
- `csvddfVersion`: ignored.

**File compression:**
Resource file(s) with `path` ending in `.gz`, `.bz2`, `.xz`, or `.zip` are automatically decompressed using default `readr::read_delim()` functionality. Only `.gz` files can be read directly from URL `paths`. Only the extension in `path` can be used to indicate compression type, the `compression` property is ignored.

**Ignored resource properties:**

- `title`
- `description`
- `format`
- `mediatype`
- `bytes`
- `hash`
- `sources`
- `licenses`

### Table schema properties

schema is required and must follow the Table Schema specification. It can either be a JSON object or a path or URL referencing a JSON object.

- Field names are used as column headers.
- Field types are use as column types (see further).
- missingValues are used to interpret as NA, with "" as default.

**Field types:**

Field type is used to set the column type, as follows:

- string as character; or factor when enum is present. format is ignored.
- number as double; or factor when enum is present. Use bareNumber: false to ignore whitespace and non-numeric characters. decimalChar (. by default) and groupChar (undefined by default) can be defined, but the most occurring value will be used as a global value for all number fields of that resource.
- integer as double (not integer, to avoid issues with big numbers); or factor when enum is present. Use bareNumber: false to ignore whitespace and non-numeric characters.
- boolean as logical. Non-default trueValues/falseValues are not supported.
- object as character.
- array as character.
- date as date. Supports format, with values default (ISO date), any (guess ymd) and Python/C strptime patterns, such as %a, %d %B %Y for Sat, 23 November 2013. %x is %m/%d/%y. %j, %U, %w and %W are not supported.
- time as hms::hms(). Supports format, with values default (ISO time), any (guess hms) and Python/C strptime patterns, such as %I%p%M:%S.%f%z for 8AM30:00.300+0200.
- datetime as POSIXct. Supports format, with values default (ISO datetime), any (ISO datetime) and the same patterns as for date and time. %c is not supported.
- year as date, with 01 for month and day.
- yearmonth as date, with 01 for day.
- duration as character. Can be parsed afterwards with lubridate::duration().
- geopoint as character.
- geojson as character.
- any as character.
- Any other value is not allowed.
- Type is guessed when not provided.

### See Also

Other read functions: read_package(), resources()

### Examples

```
# Read a datapackage.json file
package <- read_package(
  system.file("extdata", "datapackage.json", package = "frictionless")
)
```

```
# List resources
resources(package)

# Read data from the resource "observations"
read_resource(package, "observations")

# The above tibble is merged from 2 files listed in the resource path
package$resources[[2]]$path

# The column names and types are derived from the resource schema
purrr::map_chr(package$resources[[2]]$schema$fields, "name")
purrr::map_chr(package$resources[[2]]$schema$fields, "type")
```

---

`remove_resource` *Remove a Data Resource*

---

## Description

Removes a Data Resource from a Data Package, i.e. it removes one of the described `resources`.

## Usage

```
remove_resource(package, resource_name)
```

## Arguments

| | |
|---|---|
| `package` | List describing a Data Package, created with `read_package()` or `create_package()`. |
| `resource_name` | Name of the Data Resource. |

## Value

Provided package with one fewer resource.

## See Also

Other edit functions: `add_resource()`, `get_schema()`

## Examples

```
# Load the example Data Package
package <- example_package

# List resources
resources(package)

# Remove the resource "observations"
package <- remove_resource(package, "observations")

# List resources ("observations" removed)
resources(package)
```

## resources *List Data Resources*

### Description

Lists the names of the Data Resources included in a Data Package.

### Usage

```
resources(package)
```

### Arguments

| | |
|---|---|
| package | List describing a Data Package. |

### Value

Character vector with the Data Resource names.

### See Also

Other read functions: `read_package()`, `read_resource()`

### Examples

```
# Load the example Data Package
package <- example_package

# List resources
resources(package)
```

## write_package *Write a Data Package to disk*

### Description

Writes a Data Package and its related Data Resources to disk as a `datapackage.json` and CSV files. Already existing CSV files of the same name will not be overwritten. The function can also be used to download a Data Package in its entirety. The Data Resources are handled as follows:

- Resource `path` has at least one local path (e.g. `deployments.csv`): CSV files are copied or downloaded to `directory` and `path` points to new location of file(s).
- Resource `path` has only URL(s): resource stays as is.
- Resource has inline `data` originally: resource stays as is.
- Resource has inline `data` as result of adding data with `add_resource()`: data are written to a CSV file using `readr::write_csv()`, `path` points to location of file, `data` property is removed. Use `compress = TRUE` to gzip those CSV files.

## Usage

```
write_package(package, directory = ".", compress = FALSE)
```

## Arguments

| | |
|---|---|
| package | List describing a Data Package, created with read_package() or create_package(). |
| directory | Path to local directory to write files to. |
| compress | If TRUE, data of added resources will be gzip compressed before being written to disk (e.g. deployments.csv.gz). |

## Value

package as written to file (invisibly).

## Examples

```
# Load the example Data Package from disk
package <- read_package(
  system.file("extdata", "datapackage.json", package = "frictionless")
)

# List resources
resources(package)

# Write the (unchanged) Data Package to disk
write_package(package, directory = "my_directory")

# Check files
list.files("my_directory")

# No files written for the "observations" resource, since those are all URLs.
# No files written for the "media" resource, since it has inline data.

# Clean up (don't do this if you want to keep your files)
unlink("my_directory", recursive = TRUE)
```

# Index