Package: frictionless (via r-universe)

October 29, 2026

Title Read and Write Frictionless Data Packages

Version 2.0.1.9000

Date 2026-10-29

Description Read and write Frictionless Data Packages. A 'Data Package' (<https://datapackage.org/>) 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.gofair.foundation/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

Depends R (>= 4.1.0)

Imports cli, dplyr, httr, jsonlite, lifecycle, purrr, readr (>= 2.1.0), rlang, utils, yaml

Suggests hms, knitr, lubridate, rmarkdown, stringi, testthat (>= 3.0.0), tibble, withr

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.0

Config/pak/sysreqs libssl-dev libx11-dev

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

Date/Publication 2026-10-29 15:59:00 UTC

RemoteUrl https://github.com/frictionlessdata/frictionless-r

RemoteRef HEAD

RemoteSha 7e49a358c4aaf293fc427480c7b199e58351371b

Contents

add_properties . . . 2
add_resource . . . 3
check_package . . . 5
create_package . . . 6
create_schema . . . 7
example_package . . . 8
print.datapackage . . . 9
read_package . . . 9
read_resource . . . 10
remove_resource . . . 11
resource . . . 12
resource_names . . . 13
schema . . . 14
upgrade_package . . . 15
version . . . 16
write_package . . . 17

Index **19**

`add_properties` *Add properties*

Description

Adds or inserts provided arguments as elements to a list. Attributes and classes of the original list are retained.

Usage

```
add_properties(x, ..., after = length(x))
```

Arguments

`x`	A list or vector.
`...`	Elements to add, as named arguments.
`after`	Position after which elements should be added. Use `0` to add elements at the beginning.

Details

`add_properties()` can be used to add (custom) metadata properties to a Data Package, Data Resource, Table Dialect or Table Schema. Note that added properties are not validated. See the vignettes for an overview of all standard metadata properties.

`add_properties()` has some advantages over:

- `append()`: removes attributes and classes, invalidating a Data Package object.

- Direct assignment (`package$title <- "My package"`): no control over the position of the added element.

`add_resource()` also supports adding metadata properties to a Data Resource, except those automatically set by that function.

Value

`x` with the added elements.

See Also

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

Examples

```
# Add property to a generic list
list <- list(a = 1, b = 2)
add_properties(list, added = 3)

# Add properties to a Data Package
package <- create_package()
package <- add_properties(
  package,
  title = "Example package",
  keywords = c("camera traps", "frictionlessdata"),
  after = 1 # Add after the first property
)
package

str(package)
```

`add_resource` *Add a Data Resource*

Description

Adds a Data Resource to a Data Package. The resource will be a Tabular Data Resource.

Usage

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

Arguments

package	Data Package object, as returned by 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): • Data frame: attached to the resource as data and written to a CSV file when using write_package(). • One or more paths or URLs to CSV files 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().
replace	If TRUE, allows an existing resource of the same name to be replaced.
delim	Delimiter for the CSV file(s) referenced in data (e.g. \t for a tab-separated file). Will be set as delimiter in the resource Table Dialect, so read functions know how to read the file(s). Ignored if data is a data frame.
...	Additional metadata properties to add to the resource. Note that added properties are not validated and ignored by read_resource(). The following properties are automatically set and can't be provided with ...: $schema, name, path, data, type, format, mediatype, encoding, dialect and schema.

Details

See vignette("data-resource") (and to a lesser extend vignette("table-dialect")) to learn how this function implements the Data Package standard.

Value

package with one additional resource.

See Also

Other edit functions: add_properties(), remove_resource()

Examples

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

# List the resources
resource_names(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 the resource "positions" from the data frame
package <- add_resource(package, "positions", data = df)

# Add the resource "positions_with_schema", with a user-defined schema and title
my_schema <- create_schema(df)
package <- add_resource(
  package,
  resource_name = "positions_with_schema",
  data = df,
  schema = my_schema,
  title = "Positions with schema"
)

# Replace the resource "observations" with a file-based resource (2 TSV files)
path_1 <-
  system.file("extdata", "v1", "observations_1.tsv", package = "frictionless")
path_2 <-
  system.file("extdata", "v1", "observations_2.tsv", package = "frictionless")
package <- add_resource(
  package,
  resource_name = "observations",
  data = c(path_1, path_2),
  replace = TRUE,
  delim = "\t"
)

# List the resources ("positions" and "positions_with_schema" added)
resource_names(package)
```

---

check_package *Check a Data Package object*

Description

Check if an object is a Data Package object with the required properties.

Usage

```
check_package(package)
```

Arguments

package	Data Package object, as returned by `read_package()` or `create_package()`.

Value

package invisibly or an error.

Examples

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

# Check if the Data Package is valid (invisible return)
check_package(package)
```

---

create_package *Create a Data Package*

Description

Initiates a Data Package object, either from scratch or from an existing list. This Data Package object is a list with the following characteristics:

- All properties of the original `descriptor`.
- A `resources` property, set to an empty list if undefined.
- A `directory` attribute, set to `"."` for the current directory if undefined. It is used as the base path to access resources with `read_resource()`.
- A `datapackage` subclass.

Usage

```
create_package(descriptor = NULL)
```

Arguments

`descriptor`	List to be made into a Data Package object. If undefined, an empty Data Package will be created from scratch.

Details

See `vignette("data-package")` to learn how this function implements the Data Package standard. `check_package()` is automatically called on the created package to make sure it is valid.

Value

A Data Package object.

See Also

Other create functions: `create_schema()`

Examples

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

package

# See the structure of the (empty) Data Package
str(package)
```

`create_schema` *Create a Table Schema from 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.

Details

See `vignette("table-schema")` to learn how this function implements the Data Package standard.

Value

List describing a Table Schema.

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` *Read the example Data Package*

Description

Reads the example Data Package included in `frictionless`. This dataset is used in examples, vignettes, and tests and contains dummy camera trap data organized in 3 Data Resources:

1. `deployments`: one local data file referenced in `"path": "deployments.csv"`.
2. `observations`: two local data files referenced in `"path": ["observations_1.tsv", "observations_2.tsv"]`.
3. `media`: inline data stored in `data`.

Usage

```
example_package(version = "2.0")
```

Arguments

`version`	Data Package standard version number.

Details

The example Data Package is available in two versions:

- `1.0`: specified in the v1 specification.
- `2.0`: specified in the v2 specification.

Value

A Data Package object, see `create_package()`.

Examples

```
# Load the v2 example Data Package (default)
example_package()

# Load the v1 example Data Package
example_package(version = "1.0")
```

print.datapackage *Print a Data Package*

Description

Prints a human-readable summary of a Data Package, including its resources and a link to more information (if provided in `package$id`).

Usage

```
## S3 method for class 'datapackage'
print(x, ...)
```

Arguments

`x`	Data Package object, as returned by `read_package()` or `create_package()`.
`...`	Further arguments, they are ignored by this function.

Value

`print()` with a summary of the Data Package object.

Examples

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

# Print a summary of the Data Package
package # Or print(package)
```

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.

Details

See `vignette("data-package")` to learn how this function implements the Data Package standard.

Value

A Data Package object, see `create_package()`.

See Also

Other read functions: `read_resource()`

Examples

```
# Read a datapackage.json file
package <- read_package(
  system.file("extdata", "v1", "datapackage.json", package = "frictionless")
)

package

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

---

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, col_select = NULL)
```

Arguments

`package`	Data Package object, as returned by `read_package()` or `create_package()`.
`resource_name`	Name of the Data Resource.
`col_select`	Character vector of the columns to include in the result, in the order provided. Selecting columns can improve read speed.

Details

See `vignette("data-resource")`, `vignette("table-dialect")` and `vignette("table-schema")` to learn how this function implements the Data Package standard.

Value

A `tibble::tibble()` with the Data Resource's tabular data. If there are parsing problems, a warning will alert you. You can retrieve the full details by calling `problems()` on your data frame.

See Also

Other read functions: `read_package()`

Examples

```
# Read a datapackage.json file
package <- read_package(
  system.file("extdata", "v1", "datapackage.json", package = "frictionless")
)

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")

# Read data from the resource "deployments" with column selection
read_resource(package, "deployments", col_select = c("latitude", "longitude"))
```

---

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`	Data Package object, as returned by `read_package()` or `create_package()`.
`resource_name`	Name of the Data Resource.

Value

`package` with one fewer resource.

See Also

Other edit functions: add_properties(), add_resource()

Examples

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

# List the resources
resource_names(package)

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

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

---

resource *Get or overwrite a Data Resource*

Description

Gets or overwrites a Data Resource from/in a Data Package. These functions are **designed for internal use in other packages**. For public manipulation of resources, use add_resource() and remove_resource().

resource() gets a Data Resource from a Data Package by its name. The returned value will be a list describing a Data Resource, with a new attribute data_location to indicate how the data are attached. If present, path will be updated to the full path(s).
resource<- overwrites a Data Resource in Data Package with a new value. The assigned value will typically be a resource obtained with resource() and then manipulated. The assignment function will therefore revert internal changes made by resource() (i.e. removing the attribute data_location and updating paths to the original values). The assigned value is otherwise **not validated**, so use with care.

Usage

```
resource(package, resource_name)

resource(package, resource_name) <- value
```

Arguments

package	Data Package object, as returned by read_package() or create_package().
resource_name	Name of the Data Resource.
value	Value to assign.

Details

See `vignette("data-resource")` to learn more about Data Resource.

Value

List describing a Data Resource.

`package` with overwritten resource.

See Also

Other accessor functions: `resource_names()`, `schema()`

Examples

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

# Get the resource "deployments"
resource <- resource(package, "deployments")
str(resource)

# Update the resource
resource$description <- "Table with deployments."

# Overwrite the resource
resource(package, "deployments") <- resource

# Updating a resource property can also be done in one step
resource(package, "deployments")$description <- "Table with deployments."
```

`resource_names` *List Data Resource names*

Description

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

Usage

```
resource_names(package)
```

Arguments

`package`	Data Package object, as returned by `read_package()` or `create_package()`.

Value

Character vector with the Data Resource names.

See Also

Other accessor functions: `resource()`, `schema()`

Examples

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

# List the resources
resource_names(package)
```

---

schema *Get the Table Schema of a Data Resource*

Description

Gets 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

```
schema(package, resource_name)
```

Arguments

`package`	Data Package object, as returned by `read_package()` or `create_package()`.
`resource_name`	Name of the Data Resource.

Details

See `vignette("table-schema")` to learn more about Table Schema.

Value

List describing a Table Schema.

See Also

Other accessor functions: `resource()`, `resource_names()`

Examples

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

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

upgrade_package *Upgrade a Data Package to v2*

Description

Upgrades a Data Package, its Data Resources and Table Schemas to the v2 specification.

Usage

```
upgrade_package(package)
```

Arguments

package	Data Package object, as returned by `read_package()` or `create_package()`.

Value

Upgraded package.

Upgrade details

Data Package:

`upgrade_package()` upgrades a v1 descriptor to v2 as follows:

- Adds `$schema` as first property and sets it to the recommended v2 value (`"https://datapackage.org/profiles/2.0` except for certain `profile` values.
- Removes `profile`, but retains its value in `$schema` if it is a URL to a custom profile (see backwards compatibility).
- Converts `contributors` `"role": "value"` to `"roles": ["value"]` (see changelog).

Data Resource:

`upgrade_package()` upgrades any v1 resource to v2 as follows:

- Adds `$schema` as first property and sets it to the recommended v2 value (`"https://datapackage.org/profiles/2.0`
- Removes `profile`, but converts `"profile" = "tabular-data-resource"` to `"type" = "table"` (see backwards compatibility).

Table Dialect:

`upgrade_package()` leaves the dialect as is for all resources.

Table Schema:

`upgrade_package()` upgrades any verbose v1 schema to v2 as follows:

- Adds `$schema` as first property and sets it to the recommended v2 value (`"https://datapackage.org/profiles/2.0`
- Converts `primaryKey` single values to an array (see changelog).
- Converts `foreignKeys` single values in `fields` to an array and removes `reference$resource` if it is self-referential (see changelog).

Schemas referenced by path or URL are left as is.

See Also

Other versioning functions: `version()`

Examples

```
# Load the v1 example Data Package
(package <- example_package(version = "1.0"))

# Upgrade
(package_upgraded <- upgrade_package(package))

# Check version of one of the resources
version(resource(package_upgraded, "deployments"))
```

`version` *Get the specification version number*

Description

Determines what version of the Data Package, Data Resource, Table Dialect or Table Schema standard is used, based on the `$schema` property.

- `"1.0"`: v1 specification. Assumed if `$schema` is missing.
- `"2.0"`: v2 specification.
- `">=2.0"`: assumed if `$schema` is defined by deviating from the default (e.g. an extension).

Usage

```
version(x)
```

Arguments

`x`	A list describing a Data Package, Data Resource, Table Dialect or Table Schema.

Value

Data Package standard version number.

See Also

Other versioning functions: `upgrade_package()`

Examples

```
# Data Package
package <- example_package()
version(package)

# Data Resource
resource <- resource(package, "observations")
version(resource)

# Table Dialect
version(resource$dialect)

# Table Schema
schema <- schema(package, "observations")
version(schema)
```

`write_package` *Write a Data Package to disk*

Description

Writes a Data Package and its Data Resources to disk as a `datapackage.json` and CSV files.

Usage

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

Arguments

`package`	Data Package object, as returned by `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` invisibly, as written to file.

Writing data to CSV files

`write_package()` will write data to CSV files depending on how these are attached to the Data Resource:

- **Data frame**:

  ```
  add_resource(package, "media", data = df)
  ```

Data are written to a CSV file in `directory` using `readr::write_csv()`. The CSV file will have the same name as the resource, overwriting any existing file with the same name. Use `compress = TRUE` to gzip the CSV file.

- One or more **paths** to local CSV files in a **different directory**:

  ```
  add_resource(package, "media", data = "other-directory/media.csv")
  ```

  CSV files are copied to `directory`, overwriting existing files with the same name.

- One or more **paths** to local CSV files in the **same directory**:

  ```
  add_resource(package, "media", data = "directory/media.csv")
  ```

  CSV files are left as is (no overwrite). This allows you to read and write a `datapackage.json` to the same directory, without altering the CSV files of resources you did not manipulate.

- One or more **URLs** to CSV files:

  ```
  add_resource(package, "media", data = "https://example.org/media.csv")
  ```

  Files are not downloaded.

- Mix of **URLs and paths** to CSV files:

  ```
  add_resource(
    package, "media",
    data = c("https://example.org/media.csv", "media.csv")
  )
  ```

  Remote CSV files are downloaded to `directory`, overwriting existing files with the same name. Local CSV files are handled as described above.

- **Inline data**: No files are written.

In all above cases `path` is added or updated to the new file location(s) when appropriate.

Examples

```
# Load the example Data Package from disk
package <- read_package(
  system.file("extdata", "v1", "datapackage.json", package = "frictionless")
)

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

∗ **accessor functions**
  resource, 12
  resource_names, 13
  schema, 14
∗ **check functions**
  check_package, 5
∗ **create functions**
  create_package, 6
  create_schema, 7
∗ **edit functions**
  add_properties, 2
  add_resource, 3
  remove_resource, 11
∗ **print functions**
  print.datapackage, 9
∗ **read functions**
  read_package, 9
  read_resource, 10
∗ **sample data**
  example_package, 8
∗ **versioning functions**
  upgrade_package, 15
  version, 16
∗ **write functions**
  write_package, 17

add_properties, 2
add_properties(), *4*, *12*
add_resource, 3
add_resource(), *3*, *12*
append(), *2*

check_package, 5
check_package(), *6*
create_package, 6
create_package(), *4, 5*, *7–15*, *17*
create_schema, 7
create_schema(), *4*, *6*

example_package, 8

print(), *9*
print.datapackage, 9
problems(), *11*

read_package, 9
read_package(), *4, 5*, *9–15*, *17*
read_resource, 10
read_resource(), *4*, *6*, *10*
readr::read_delim(), *4*, *10*
readr::write_csv(), *18*
remove_resource, 11
remove_resource(), *3, 4*, *12*
resource, 12
resource(), *14*
resource<- *(*resource*)*, 12
resource_names, 13
resource_names(), *13, 14*

schema, 14
schema(), *13, 14*

tibble::tibble(), *11*

upgrade_package, 15
upgrade_package(), *16*

version, 16
version(), *16*

write_package, 17
write_package(), *4*