

Package ‘frictionless’

July 31, 2026

Title Read and Write Frictionless Data Packages

Version 1.3.0

Date 2026-07-30

Description Read and write Frictionless Data Packages. A 'Data Package' ([\(<https://specs.frictionlessdata.io/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/>](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>

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

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Peter Desmet [aut, cre] (ORCID: [<https://orcid.org/0000-0002-8442-8025>](https://orcid.org/0000-0002-8442-8025), affiliation: Research Institute for Nature and Forest (INBO)),
Damiano Oldoni [aut] (ORCID: [<https://orcid.org/0000-0003-3445-7562>](https://orcid.org/0000-0003-3445-7562), affiliation: Research Institute for Nature and Forest (INBO)),
Pieter Huybrechts [aut] (ORCID: [<https://orcid.org/0000-0002-6658-6062>](https://orcid.org/0000-0002-6658-6062), affiliation: Research Institute for Nature and Forest (INBO)),

Sanne Govaert [aut] (ORCID: <<https://orcid.org/0000-0002-8939-1305>>, affiliation: Research Institute for Nature and Forest (INBO)),
Kyle Husmann [ctb] (ORCID: <<https://orcid.org/0000-0001-9875-8976>>, affiliation: Pennsylvania State University),
Research Institute for Nature and Forest (INBO) [cph] (ROR: <<https://ror.org/00j54wy13>>),
Research Foundation - Flanders [fnd] (<https://lifewatch.be>),
Beatriz Milz [rev] (ORCID: <<https://orcid.org/0000-0002-3064-4486>>),
João Martins [rev] (ORCID: <<https://orcid.org/0000-0001-7961-4280>>)

Maintainer Peter Desmet <peter.desmet@inbo.be>

Repository CRAN

Date/Publication 2026-07-31 06:30:03 UTC

Contents

add_resource	2
check_package	4
create_package	5
create_schema	6
example_package	7
print.datapackage	8
read_package	9
read_resource	10
remove_resource	11
resource	12
resource_names	13
schema	14
version	15
write_package	15

Index	18
--------------	-----------

add_resource	<i>Add a Data Resource</i>
--------------	----------------------------

Description

Adds a 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,
    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, e.g. title = "My title", validated = FALSE. These are not verified against specifications and are ignored by read_resource() . The following properties are automatically set and can't be provided with ...: name, data, path, schema, profile, format, mediatype, encoding and dialect.

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: [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	<i>Create a Data Package</i>
----------------	------------------------------

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	<i>Create a Table Schema for a data frame</i>
---------------	---

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	<i>Read the example Data Package</i>
-----------------	--------------------------------------

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 = "1.0")
```

Arguments

version	Data Package standard version.
---------	--------------------------------

Details

The example Data Package is available in two versions:

- 1.0: specified as a **Data Package v1**.
- 2.0: specified as a **Data Package v2**.

Value

A Data Package object, see [create_package\(\)](#).

Examples

```
# Version 1
example_package()

# Version 2
example_package(version = "2.0")
```

print.datapackage	<i>Print a Data Package</i>
-------------------	-----------------------------

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	<i>Read a Data Package descriptor file (datapackage.json)</i>
--------------	---

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 <code>datapackage.json</code> 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	<i>Read data from a Data Resource into a tibble data frame</i>
---------------	--

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 <code>read_package()</code> or <code>create_package()</code> .
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	<i>Remove a Data Resource</i>
-----------------	-------------------------------

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_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

<code>package</code>	Data Package object, as returned by read_package() or create_package() .
<code>resource_name</code>	Name of the Data Resource.
<code>value</code>	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	<i>List Data Resource names</i>
----------------	---------------------------------

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)
```

version	<i>Get Data Package version</i>
---------	---------------------------------

Description

Determines what version of the **Data Package standard** is used by a Data Package, based on the **\$schema** property. Version "1.0" is assumed if \$schema is missing, version ">=2.0" is assumed for custom values (e.g. **extensions**).

Usage

```
version(package)
```

Arguments

package Data Package object, as returned by [read_package\(\)](#) or [create_package\(\)](#).

Value

Data Package version number (e.g. "1.0").

Examples

```
package <- example_package()
version(package)
```

write_package	<i>Write a Data Package to disk</i>
---------------	-------------------------------------

Description

Writes a Data Package and its related 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, [4](#)
- * **create functions**
 - create_package, [5](#)
 - create_schema, [6](#)
- * **edit functions**
 - add_resource, [2](#)
 - remove_resource, [11](#)
- * **print functions**
 - print.datapackage, [8](#)
- * **read functions**
 - read_package, [9](#)
 - read_resource, [10](#)
- * **sample data**
 - example_package, [7](#)
- * **version functions**
 - version, [15](#)
- * **write functions**
 - write_package, [15](#)

add_resource, [2](#)
add_resource(), [11](#), [12](#)

check_package, [4](#)
check_package(), [5](#)
create_package, [5](#)
create_package(), [3](#), [5–15](#)
create_schema, [6](#)
create_schema(), [3](#), [6](#)

example_package, [7](#)

print(), [8](#)
print.datapackage, [8](#)
problems(), [10](#)

read_package, [9](#)
read_package(), [3](#), [5](#), [8](#), [10–15](#)
read_resource, [10](#)
read_resource(), [3](#), [5](#), [9](#)
readr::read_delim(), [3](#), [10](#)
readr::write_csv(), [16](#)
remove_resource, [11](#)
remove_resource(), [3](#), [12](#)
resource, [12](#)
resource(), [13](#), [14](#)
resource<- (resource), [12](#)
resource_names, [13](#)
resource_names(), [12](#), [14](#)

schema, [14](#)
schema(), [12](#), [13](#)

tibble::tibble(), [10](#)

version, [15](#)

write_package, [15](#)
write_package(), [3](#)