

# Package ‘datapackage’

May 8, 2026

**Type** Package

**Title** Creating and Reading Data Packages

**Version** 0.2.3

**Description** Open, read data from and modify Data Packages. Data Packages are an open standard for bundling and describing data sets (<https://datapackage.org>). When data is read from a Data Package care is taken to convert the data as much as possible to R appropriate data types. The package can be extended with plugins for additional data types.

**BugReports** <https://github.com/djvanderlaan/datapackage/issues>

**URL** <https://github.com/djvanderlaan/datapackage>

**Depends** R (>= 4.1.0)

**Imports** yaml, jsonlite, iso8601, utils, tools, methods

**Suggests** simplermardown, data.table, codelist, LaF

**VignetteBuilder** simplermardown

**License** GPL-3

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**NeedsCompilation** no

**Author** Jan van der Laan [aut, cre] (ORCID:  
<<https://orcid.org/0000-0002-0693-1514>>)

**Maintainer** Jan van der Laan <r@eoos.dds.nl>

**Repository** CRAN

**Date/Publication** 2025-05-28 06:30:02 UTC

## Contents

csv_reader . . . . .	3
csv_writer . . . . .	4
dp_add_reader . . . . .	4

dp_add_writer . . . . .	5
dp_apply_schema . . . . .	6
dp_categorieslist . . . . .	7
dp_check_dataresource . . . . .	8
dp_check_field . . . . .	9
dp_field . . . . .	10
dp_field_names . . . . .	10
dp_generate_dataresource . . . . .	11
dp_generate_fielddescriptor . . . . .	12
dp_get_connection . . . . .	14
dp_get_data . . . . .	15
dp_get_datapackage . . . . .	16
dp_load_from_datapackage . . . . .	16
dp_nresources . . . . .	17
dp_properties . . . . .	17
dp_property . . . . .	18
dp_resource . . . . .	19
dp_resources<- . . . . .	20
dp_resource_names . . . . .	21
dp_save_as_datapackage . . . . .	21
dp_to_boolean . . . . .	22
dp_to_code . . . . .	23
dp_to_date . . . . .	24
dp_to_datetime . . . . .	24
dp_to_factor . . . . .	25
dp_to_integer . . . . .	26
dp_to_number . . . . .	27
dp_to_string . . . . .	27
dp_to_time . . . . .	28
dp_to_year . . . . .	29
dp_to_yearmonth . . . . .	29
dp_write_data . . . . .	30
fwf_reader . . . . .	31
new_contributor . . . . .	32
new_datapackage . . . . .	33
new_dataresource . . . . .	34
open_datapackage . . . . .	35
PropertiesDatapackage . . . . .	36
PropertiesDataresource . . . . .	38
PropertiesFielddescriptor . . . . .	41

---

`csv_reader`*Read the CSV-data for a Data Resource*

---

## Description

Read the CSV-data for a Data Resource

## Usage

```
csv_reader(  
  path,  
  resource,  
  use_fread = FALSE,  
  convert_categories = c("no", "to_factor"),  
  as_connection = FALSE,  
  ...  
)
```

## Arguments

<code>path</code>	path to the data set.
<code>resource</code>	a Data Resource.
<code>use_fread</code>	use the <a href="#">fread</a> function instead of <a href="#">read.csv</a> and return a <code>data.table</code> .
<code>convert_categories</code>	how to handle columns for which the field descriptor has a <code>categories</code> property. Passed on to <a href="#">dp_apply_schema</a> .
<code>as_connection</code>	This argument is ignored. The function will always return a <code>data.frame</code> .
<code>...</code>	additional arguments are passed on to <a href="#">read.csv</a> or <a href="#">fread</a> . Note that some arguments are already set by <code>csv_reader</code> , so not all arguments are available to use as additional arguments.

## Value

Returns a `data.frame` with the data.

## See Also

Generally used by calling [dp\\_get\\_data](#).

---

csv_writer	<i>Write data of data resource to CSV-file</i>
------------	--

---

**Description**

Write data of data resource to CSV-file

**Usage**

```
csv_writer(x, resource_name, datapackage, use_fwrite = FALSE, ...)
```

**Arguments**

x	data.frame with the data to write
resource_name	name of the data resource in the data package.
datapackage	the Data Package to which the file should be written.
use_fwrite	write the file using fwrite from the data.table package.
...	ignored for now

**Value**

The function doesn't return anything. It is called for its side effect of creating CSV-files in the directory of the data package.

---

dp_add_reader	<i>Add a reader function for a specific format</i>
---------------	--

---

**Description**

Add a reader function for a specific format

**Usage**

```
dp_add_reader(
  format,
  reader,
  mediatypes = character(),
  extensions = character()
)
```

**Arguments**

format	the data format read by the reader. Should be a length 1 character vector.
reader	the reader function. See details.
mediatypes	a character vector with the media-types that are used for the format.
extensions	a character vector with typical file extensions used by the format.

**Details**

Adds a reader for a given format. The reader is added to a list of reader references by the format. It is also possible to assign mediatypes and file extensions to the format. When the format for a given Data Resource is missing, `dp_get_data` will first check if a mediatype is associated with the resource and will try to look up which format belongs to the given mediatype. If that doesn't result in a valid format, `dp_get_data` will try the same with the extension of the file.

Note that adding a reader for an existing format will overwrite the existing reader.

**Value**

Does not return anything (`invisible(NULL)`).

**Examples**

```
# Add a very simple reader for json
json_reader <- function(path, resource, ...) {
  lapply(path, function(fn) {
    jsonlite::read_json(fn)
  })
}

dp_add_reader("json", json_reader, c("application/json"), "json")
```

---

dp\_add\_writer

*Add a writer function for a specific format*


---

**Description**

Add a writer function for a specific format

**Usage**

```
dp_add_writer(format, writer)
```

**Arguments**

format	the data format read by the writer Should be a length 1 character vector.
writer	the writer function. See details.

**Details**

Adds a writer for a given format. The writer is added to a list of writers referenced by the format. The writer function should accept 'data' with the data as its first argument, 'resource\_name' the name of the resource to which the data set belongs, 'datapackage' that datapackage to which the data should be written.

Note that adding a writer for an existing format will overwrite the existing writer

**Value**

Does not return anything (`invisible(NULL)`).

**Examples**

```
# Add a very simple writer for json
json_writer <- function(data, resource_name, datapackage, ...) {
  dataresource <- dp_resource(datapackage, resource_name)
  path <- dp_path(dataresource, full_path = TRUE)
  jsonlite::write_json(data, path)
}

dp_add_writer("json", json_writer)
```

---

dp_apply_schema	<i>Convert columns of data.frame to their correct types using table schema</i>
-----------------	--

---

**Description**

Convert columns of data.frame to their correct types using table schema

**Usage**

```
dp_apply_schema(
  dta,
  resource,
  convert_categories = c("no", "to_factor", "to_code"),
  reorder = TRUE,
  ...
)
```

**Arguments**

dta	a data.frame or data.table.
resource	an object with the Data Resource of the data set.
convert_categories	how to handle columns for which the field descriptor has a categories property. This should either be the strings "no", "to_factor", "to_code", the name of a function or a function. When equal to "no" the field is returned as is; when equal to "to_factor" each column is transformed using <code>dp_to_factor</code> ; when equal to "to_code" each column is transformed using <code>dp_to_code</code> . In other cases the function is called with the column as its first parameter and <code>warn = FALSE</code> as its second argument. The result of this function call is added to the resulting data set.
reorder	put the columns in the same order as the fields in the data resource.
...	additional arguments are passed on to the <code>dp_to_&lt;fieldtype&gt;</code> functions (e.g. <code>dp_to_number</code> ).

## Details

Converts each column in `dta` to the correct R-type using the type information in the table schema. For example, if the original column type in `dta` is a character vector and the table schema specifies that the field is of type number, the column is converted to numeric using the decimal separator and thousands separator specified in the field descriptor (or default values for these if not).

## Value

Returns a copy of the input `data.frame` with columns modified to match the types given in the table schema. When `reorder = TRUE` columns are put in the same order as in the data resource with fields not in the data resource put at the back of the `data.frame`.

## See Also

This function calls conversion functions for each of the columns, see [dp\\_to\\_number](#), [dp\\_to\\_boolean](#), [dp\\_to\\_integer](#), [dp\\_to\\_date](#), [dp\\_to\\_datetime](#), [dp\\_to\\_yearmonth](#), and [dp\\_to\\_string](#).

---

<code>dp_categorieslist</code>	<i>Get the a data.frame with the categories for a variable.</i>
--------------------------------	---

---

## Description

Get the a `data.frame` with the categories for a variable.

## Usage

```
dp_categorieslist(x, ...)  
  
## Default S3 method:  
dp_categorieslist(  
  x,  
  fielddescriptor = attr(x, "fielddescriptor"),  
  datapackage = dp_get_datapackage(fielddescriptor),  
  ...  
)  
  
## S3 method for class 'fielddescriptor'  
dp_categorieslist(  
  x,  
  datapackage = dp_get_datapackage(x),  
  normalised = FALSE,  
  ...  
)
```

**Arguments**

x	the variable for which to get the Categories List
...	used to pass extra arguments on to other methods.
fielddescriptor	the Field Descriptor associated with the variable.
datapackage	the Data Package where the variable is from.
normalised	if TRUE the column with values will be named value and the the columnd with labels label.

**Value**

Returns a `data.frame` with the categories or `NULL` when none could be found.

---

`dp_check_dataresource` *Check if a data set is valid given a Data Resource*

---

**Description**

Check if a data set is valid given a Data Resource

**Usage**

```
dp_check_dataresource(
  x,
  dataresource = attr(x, "resource"),
  constraints = TRUE,
  throw = FALSE,
  tolerance = sqrt(.Machine$double.eps)
)
```

**Arguments**

x	<code>data.frame</code> to check
dataresource	dataresource object to check x against.
constraints	also check relevant constraints in the field descriptor.
throw	generate an error if the data set is not valid according to the dataresource.
tolerance	numerical tolerance used in some of the tests

**Value**

Returns `TRUE` when the field is valid. Returns a character vector with length  $\geq 1$  if the field is not valid. The text in the character values indicates why the field is not valid.

When `throw = TRUE` the function will generate an error instead of returning a character vector. When the dataset is valid the function returns `TRUE` invisibly.

**See Also**

Use `isTRUE` to check if the test was successful. See `dp_check_field` for a function that checks a column or vector.

---

dp_check_field	<i>Check if a vector is valid given a field descriptor</i>
----------------	--

---

**Description**

Check if a vector is valid given a field descriptor

**Usage**

```
dp_check_field(  
  x,  
  fielddescriptor,  
  constraints = TRUE,  
  tolerance = sqrt(.Machine$double.eps)  
)
```

**Arguments**

x	vector to test
fielddescriptor	field descriptor to test the vector against
constraints	also check relevant constraints in the field descriptor.
tolerance	numerical tolerance used in some of the tests

**Value**

Returns TRUE when the field is valid. Returns a character vector with length  $\geq 1$  if the field is not valid. The text in the character values indicates why the field is not valid.

**See Also**

Use `isTRUE` to check if the test was successful. See `dp_check_dataresource` for a function that checks a complete data set.

dp\_field                      *Get the Field Descriptor associated with a certain field in a Data Resource*

---

**Description**

Get the Field Descriptor associated with a certain field in a Data Resource

**Usage**

```
dp_field(x, field_name)
```

**Arguments**

x                              a dataresource or tableschema object.  
field\_name                    length one character vector with the name of the field.

**Value**

An object of type fielddescriptor.

---

dp\_field\_names                *List the fields in a Data Resource*

---

**Description**

List the fields in a Data Resource

**Usage**

```
dp_field_names(x)
```

**Arguments**

x                              object for which to get the field names. This can either be a Data Resource or Table Schema.

**Value**

Returns a character vector with the fields in the Data Resource.

---

 dp\_generate\_dataresource

*Generate Data Resource for a dataset*


---

### Description

Generate Data Resource for a dataset

### Usage

```
dp_generate_dataresource(
  x,
  name,
  path = paste0(name, getextension(format)),
  format = "csv",
  mediatype = getmediatype(format),
  use_existing = FALSE,
  categories_type = c("regular", "resource"),
  categorieslist = iscategoriestype(x),
  ...
)
```

### Arguments

x	data.frame for which to generate the Data Resources.
name	name of the Data Resource
path	name of the file in which to store the dataset. This should be a path relative to the location of the directory in which the Data Package in which the Data Resource will be stored.
format	the data format in which the data is stored.
mediatype	mediatype of the data
use_existing	use existing field descriptors if present.
categories_type	how should categories be stored. See <a href="#">dp_generate_fielddescriptor</a> .
categorieslist	does the data resource function as a categories list for fields in another data resource
...	Currently ignored

### Value

Returns a Data Resource object.

Note that this function does not create the file at path. The export of the Data Resource is automatically set to CSV.

## Examples

```
# generate an example dataset
dta <- data.frame(a = 1:3, b = factor(letters[1:3]))
resource <- dp_generate_dataresource(dta, "example")
print(resource)
```

---

dp\_generate\_fielddescriptor

*Generate a fielddescriptor for a given variable in a dataset*

---

## Description

Generate a fielddescriptor for a given variable in a dataset

## Usage

```
dp_generate_fielddescriptor(x, name, ...)

## Default S3 method:
dp_generate_fielddescriptor(x, name, ...)

## S3 method for class 'numeric'
dp_generate_fielddescriptor(
  x,
  name,
  use_existing = TRUE,
  use_categories = TRUE,
  categories_type = c("regular", "resource"),
  ...
)

## S3 method for class 'integer'
dp_generate_fielddescriptor(
  x,
  name,
  use_existing = TRUE,
  use_categories = TRUE,
  categories_type = c("regular", "resource"),
  ...
)

## S3 method for class 'logical'
dp_generate_fielddescriptor(
  x,
  name,
  use_existing = TRUE,
```

```
    use_categories = TRUE,
    categories_type = c("regular", "resource"),
    ...
)

## S3 method for class 'Date'
dp_generate_fielddescriptor(
  x,
  name,
  use_existing = TRUE,
  use_categories = TRUE,
  categories_type = c("regular", "resource"),
  ...
)

## S3 method for class 'character'
dp_generate_fielddescriptor(
  x,
  name,
  use_existing = TRUE,
  use_categories = TRUE,
  categories_type = c("regular", "resource"),
  ...
)

## S3 method for class 'factor'
dp_generate_fielddescriptor(
  x,
  name,
  use_existing = TRUE,
  use_categories = TRUE,
  categories_type = c("regular", "resource"),
  ...
)

## S3 method for class 'code'
dp_generate_fielddescriptor(
  x,
  name,
  use_existing = TRUE,
  use_categories = TRUE,
  categories_type = c("regular", "resource"),
  ...
)
```

**Arguments**

x                      vector for which to generate the fielddescriptor

name	name of the field in the dataset.
...	used to pass extra arguments to methods.
use_existing	use existing field descriptor if present (assumes this is stored in the 'fielddescriptor' attribute).
use_categories	do not generate a categories field except when x is a factor.
categories_type	how should categories be stored. Note that type "resource" is not officially part of the standard.

**Value**

Returns a fielddescriptor.

---

dp\_get\_connection      *Get a connection to the data belonging to a Data Resource*

---

**Description**

Get a connection to the data belonging to a Data Resource

**Usage**

```
dp_get_connection(x, ...)
```

**Arguments**

x	Can either be a Data Resource or Data Package.
...	Extra arguments are passed on to <a href="#">dp_get_data</a> .

**Details**

When x is a Data Package a additional argument resource\_name is needed to identify the correct Data Resource. See [dp\\_get\\_data](#).

This function calls dp\_get\_data with an additional as\_connection = TRUE) argument.

**Value**

Depending on the type of Data Resource a connection to the data is returned. Some readers will return the data set as a data.frame.

---

`dp_get_data`*Get the data belonging to a Data Resource*

---

### Description

Get the data belonging to a Data Resource

### Usage

```
dp_get_data(x, ..., as_connection = FALSE)

## S3 method for class 'dataresource'
dp_get_data(x, reader = "guess", ..., as_connection = FALSE)

## S3 method for class 'datapackage'
dp_get_data(x, resource_name, reader = "guess", ..., as_connection = FALSE)
```

### Arguments

<code>x</code>	a dataresource or datapackage object.
<code>...</code>	passed on to the reader
<code>as_connection</code>	Try to return a connection to the data instead of the data itself. When a reader does not support returning connections it will return the data.
<code>reader</code>	the reader to use to read the data. This should be either a function accepting the path to the data set (a character vector with possibly multiple filenames) and the Data Resource as second argument, or the character string "guess".
<code>resource_name</code>	the name of the dataresource.

### Details

When `reader = "guess"` the function will try to guess which reader to use based on the format and mediatype of the Data Resource. Currently only CSV is supported. For other data types a custom reader has to be provided unless the data is stored inside the Data Resource object.

It is also possible to assign default readers for data formats. For that see [dp\\_add\\_reader](#).

### Value

Will return the data. This will generally be a `data.frame` but depending on the file type can also be other types of R-objects.

When called with the `as_connection = TRUE` argument, it will try to return a connection to the data. This depends on the reader. When the reader does not support returning connections it will return the data.

### See Also

`dp_get_connection` is a wrapper around `dp_get_data(..., as_connection = TRUE)`.

---

dp\_get\_datapackage      *Get the Data Package associated with the object*

---

**Description**

Get the Data Package associated with the object

**Usage**

```
dp_get_datapackage(x)
```

**Arguments**

x                      the object for which to determine the associated Data Package

**Details**

This method can, of course, only determine the Data Package when this information is stored in one of the attributes of the object. This can be either be a datapackage attribute or an dataresource attribute.

**Value**

Returns a Data Resource object, or returns NULL when none could be found.

---

dp\_load\_from\_datapackage  
*Quickly read a dataset from a Data Package*

---

**Description**

Quickly read a dataset from a Data Package

**Usage**

```
dp_load_from_datapackage(path, resource_name, ...)
```

**Arguments**

path                      the directory with the Data Package  
resource\_name            the name of the Data Resource. When omitted the Data Resource with the same name as the Data Package is read in and when no such resource exists the first Data Resource is read in.  
...                        passed on to [dp\\_get\\_data](#).

**Details**

This function is a wrapper around [open\\_datapackage](#) and [dp\\_get\\_data](#). It offers a quick way to read in a dataset from a Data Package.

**Value**

Returns a dataset. Usually a `data.frame`.

---

dp_nresources	<i>Return the number of resources in a Data Package</i>
---------------	---

---

**Description**

Return the number of resources in a Data Package

**Usage**

```
dp_nresources(dp)
```

**Arguments**

dp                    A Data Package object.

**Value**

Returns an integer with the number of resources in the Data Package.

---

dp_properties	<i>Get a list of properties defined for the object</i>
---------------	--

---

**Description**

Get a list of properties defined for the object

**Usage**

```
dp_properties(x)

## S3 method for class 'readonlydatapackage'
dp_properties(x)

## S3 method for class 'editabledatapackage'
dp_properties(x)

## S3 method for class 'dataresource'
```

```
dp_properties(x)

## S3 method for class 'tableschema'
dp_properties(x)

## S3 method for class 'fielddescriptor'
dp_properties(x)
```

### Arguments

x                    the object for which to obtain the properties

### Value

Returns a character vector (possibly zero length) with the names of the properties.

### See Also

The [dp\\_property](#) method can be used to get the values of the properties.

---

dp_property	<i>Get and set properties of Data Packages and Data Resources</i>
-------------	---

---

### Description

Get and set properties of Data Packages and Data Resources

### Usage

```
dp_property(x, attribute)

## S3 method for class 'readonlydatapackage'
dp_property(x, attribute)

## S3 method for class 'editabledatapackage'
dp_property(x, attribute)

dp_property(x, attribute) <- value

## S3 replacement method for class 'readonlydatapackage'
dp_property(x, attribute) <- value

## S3 replacement method for class 'editabledatapackage'
dp_property(x, attribute) <- value

## S3 method for class 'dataresource'
dp_property(x, attribute)
```

```

## S3 replacement method for class 'dataresource'
dp_property(x, attribute) <- value

## S3 method for class 'tableschema'
dp_property(x, attribute)

## S3 replacement method for class 'tableschema'
dp_property(x, attribute) <- value

## S3 method for class 'fielddescriptor'
dp_property(x, attribute)

## S3 replacement method for class 'fielddescriptor'
dp_property(x, attribute) <- value

```

### Arguments

x	a datapackage or dataresource object.
attribute	a length 1 character vector with the name of the property.
value	the new value of the property.

### Value

Either returns the property or modifies the object.

### See Also

See [dp\\_name](#) etc. for methods for specific properties for Data Packages and [dp\\_encoding](#) etc. for specific properties for Data Resources. These specific methods also check if the input is valid for the given property.

---

dp\_resource

*Modifying the resources of a Data Package*

---

### Description

Modifying the resources of a Data Package

### Usage

```

dp_resource(x, resource_name)

## S3 method for class 'datapackage'
dp_resource(x, resource_name)

dp_resource(x, resource_name) <- value

```

```
## S3 replacement method for class 'readonlydatapackage'
dp_resource(x, resource_name) <- value

## S3 replacement method for class 'editabledatapackage'
dp_resource(x, resource_name) <- value
```

### Arguments

x                    a datapackage object.  
 resource\_name    the name of a resource.  
 value              a dataresource object.

### Details

When a resource with the name already exists this resource is overwritten. Therefore, the assignment operator can also be used to modify existing resources.

### Value

Either returns a Data Resource object or modifies the Data Package.

---

dp\_resources<-                    *Modify a set of Data Resources in a Data Package*

---

### Description

Modify a set of Data Resources in a Data Package

### Usage

```
dp_resources(x) <- value
```

### Arguments

x                    a datapackage object  
 value              a dataresource object or a list of dataresource objects .

### Value

Returns a modified x.

---

dp\_resource\_names      *Get the names of the resources in a Data Package*

---

**Description**

Get the names of the resources in a Data Package

**Usage**

```
dp_resource_names(dp)
```

**Arguments**

dp                      A datapackage object.

**Value**

Returns a character vector with the names of the data resources in the Data Package.

---

dp\_save\_as\_datapackage      *Save a dataset as a Data Package*

---

**Description**

Save a dataset as a Data Package

**Usage**

```
dp_save_as_datapackage(  
  data,  
  path,  
  name,  
  categories_type = c("regular", "resource")  
)
```

**Arguments**

data                      the data.frame with the data to save  
path                      directory in which to create the datapackage  
name                      name of the Data Resource. When omitted a name is generated.  
categories\_type            how should categories be stored. See [dp\\_generate\\_fielddescriptor](#).

**Details**

This function is a wrapper function around [new\\_datapackage](#), [dp\\_generate\\_dataresource](#) and [dp\\_write\\_data](#). These functions are called with the default arguments. This allows for a quick way to save a data set with any necessary data needed to read the dataset.

**Value**

Does not return anything. Called for the side effect of creating a directory and creating a number of files in this directory. Together these form a complete Data Package.

---

dp_to_boolean	<i>Convert a vector to 'boolean' using the specified field descriptor</i>
---------------	---

---

**Description**

Convert a vector to 'boolean' using the specified field descriptor

**Usage**

```
dp_to_boolean(x, fielddescriptor = list(), ...)
```

**Arguments**

x	the vector to convert.
fielddescriptor	the field descriptor for the field.
...	passed on to other methods.

**Details**

When fielddescriptor is missing a default field descriptor is generated.

**Value**

Will return an logical vector with fielddescriptor added as the 'fielddescriptor' attribute.

---

dp_to_code	<i>Recode a variable to code using the associated categories</i>
------------	--

---

## Description

Recode a variable to code using the associated categories

## Usage

```
dp_to_code(x, categorieslist = dp_categorieslist(x), ..., warn = FALSE)
```

## Arguments

x	the variable to recode
categorieslist	a data.frame with the categories as a data.frame.
...	passed on to <a href="#">as.codelist</a> .
warn	give a warning when there is no code list.

## Details

Uses the [code](#) method from the 'codelist' package. This package therefore needs to be installed. See the documentation of that package for how to work with 'code' objects.

By setting the option 'DP\_TRIM\_CODE' to TRUE white space at the beginning and end of character values will be removed from the category values and from x. Also multiple hyphens at the beginning of character values will be removed. This can be disabled by setting the option 'DP\_TRIM\_HYPHEN' to FALSE.

## Value

Returns a 'code' object or x when no categories could be found (categorieslist = NULL).

## See Also

An alternative is the [dp\\_to\\_factor](#) function to convert to regular R factor.

## Examples

```
fn <- system.file("examples/iris", package = "datapackage")
dp <- open_datapackage(fn)
dta <- dp |> dp_get_data("complex", convert_categories = "no")
dp_to_code(dta$factor1)

dp |> dp_get_data("complex", convert_categories = "dp_to_code")
```

---

dp\_to\_date                      *Convert a vector to 'date' using the specified field descriptor*

---

### Description

Convert a vector to 'date' using the specified field descriptor

### Usage

```
dp_to_date(x, fielddescriptor = list(), ...)
```

### Arguments

x                      the vector to convert.  
 fielddescriptor                      the field descriptor for the field.  
 ...                      passed on to other methods.

### Details

When fielddescriptor is missing a default field descriptor is generated.

### Value

Will return an Date vector with fielddescriptor added as the 'fielddescriptor' attribute.

---

dp\_to\_datetime                      *Convert a vector to 'datetime' using the specified field descriptor*

---

### Description

Convert a vector to 'datetime' using the specified field descriptor

### Usage

```
dp_to_datetime(x, fielddescriptor = list(), ...)
```

### Arguments

x                      the vector to convert.  
 fielddescriptor                      the field descriptor for the field.  
 ...                      passed on to other methods.

**Details**

When `fielddescriptor` is missing a default field descriptor is generated.

For the default format `'iso8601::iso8601todatetime'` is used to convert. This function allows more formats than the Data Package standard prescribes. When format equals "any" the default `'as.POSIXct'` function is used.

When `x` is numeric or integer, it is assumed that these are seconds since the unix time epoch (1970-01-01T00:00:00).

**Value**

Will return an `POSIXct` vector with `fielddescriptor` added as the `'fielddescriptor'` attribute.

---

<code>dp_to_factor</code>	<i>Recode a variable to factor using the associated categories</i>
---------------------------	--

---

**Description**

Recode a variable to factor using the associated categories

**Usage**

```
dp_to_factor(x, categorieslist = dp_categorieslist(x), warn = TRUE)
```

**Arguments**

`x` the variable to recode

`categorieslist` a `data.frame` with the categories as a `data.frame`.

`warn` give a warning when there is no code list.

**Details**

By setting the option `'DP_TRIM_CODE'` to `TRUE` white space at the beginning and end of character values will be ignored when comparing `x` and the category values. Also multiple hyphens at the beginning character values will be ignored. This can be disabled by setting the option `'DP_TRIM_HYPHEN'` to `FALSE`.

**Value**

Returns a factor vector or `x` when no categories could be found (`categorieslist = NULL`).

**See Also**

An alternative is the [dp\\_to\\_code](#) function to convert to `'code'` object from the `'codelist'` package.

## Examples

```
fn <- system.file("examples/iris", package = "datapackage")
dp <- open_datapackage(fn)
dta <- dp |> dp_get_data("complex", convert_categories = "no")
dp_to_factor(dta$factor1)

dp |> dp_get_data("complex", convert_categories = "to_factor")
```

---

dp\_to\_integer                      *Convert a vector to 'integer' using the specified field descriptor*

---

## Description

Convert a vector to 'integer' using the specified field descriptor

## Usage

```
dp_to_integer(x, fielddescriptor = list(), ...)
```

## Arguments

x                      the vector to convert.  
fielddescriptor                      the field descriptor for the field.  
...                      passed on to other methods.

## Details

When fielddescriptor is missing a default field descriptor is generated.

## Value

Will return an integer vector with fielddescriptor added as the 'fielddescriptor' attribute.

---

dp_to_number	<i>Convert a vector to 'number' using the specified field descriptor</i>
--------------	--

---

**Description**

Convert a vector to 'number' using the specified field descriptor

**Usage**

```
dp_to_number(x, fielddescriptor = list(), decimalChar = ".", ...)
```

**Arguments**

x	the vector to convert.
fielddescriptor	the field descriptor for the field.
decimalChar	decimal separator. Used when the field field descriptor does not specify a decimal separator.
...	passed on to other methods.

**Details**

When fielddescriptor is missing a default field descriptor is generated.

**Value**

Will return an numeric vector with fielddescriptor added as the 'fielddescriptor' attribute.

---

dp_to_string	<i>Convert a vector to 'string' using the specified fielddescriptor</i>
--------------	---

---

**Description**

Convert a vector to 'string' using the specified fielddescriptor

**Usage**

```
dp_to_string(x, fielddescriptor = list(), ...)
```

**Arguments**

x	the vector to convert.
fielddescriptor	the field descriptor for the field.
...	passed on to other methods.

**Details**

When `fielddescriptor` is missing a default field descriptor is generated.

**Value**

Will return an character vector with `fielddescriptor` added as the `'fielddescriptor'` attribute.

---

<code>dp_to_time</code>	<i>Convert a vector to 'time' using the specified field descriptor</i>
-------------------------	--

---

**Description**

Convert a vector to `'time'` using the specified field descriptor

**Usage**

```
dp_to_time(x, fielddescriptor = list(), ...)
```

**Arguments**

<code>x</code>	the vector to convert.
<code>fielddescriptor</code>	the field descriptor for the field.
<code>...</code>	passed on to other methods.

**Details**

When `fielddescriptor` is missing a default field descriptor is generated.

For the default format `'iso8601::iso8601totime'` is used to convert. This function allows more formats than the Data Package standard prescribes. When format equals `"any"` the default `'as.POSIXct'` function is used.

When `x` is numeric or integer, it is assumed that these are seconds since the unix time epoch (1970-01-01T00:00:00Z).

**Value**

Will return an Time vector (see [iso8601totime](#) with `fielddescriptor` added as the `'fielddescriptor'` attribute).

---

dp_to_year	<i>Convert a vector to 'year' using the specified field descriptor</i>
------------	--

---

**Description**

Convert a vector to 'year' using the specified field descriptor

**Usage**

```
dp_to_year(x, fielddescriptor = list(), ...)
```

**Arguments**

x	the vector to convert.
fielddescriptor	the field descriptor for the field.
...	passed on to other methods.

**Details**

When fielddescriptor is missing a default field descriptor is generated.

**Value**

Will return an integer vector with fielddescriptor added as the 'fielldescriptor' attribute.

---

dp_to_yearmonth	<i>Convert a vector to 'yearmonth' using the specified field descriptor</i>
-----------------	---

---

**Description**

Convert a vector to 'yearmonth' using the specified field descriptor

**Usage**

```
dp_to_yearmonth(x, fielddescriptor = list(), ...)
```

**Arguments**

x	the vector to convert.
fielddescriptor	the field descriptor for the field.
...	passed on to other methods.

**Details**

When `fielddescriptor` is missing a default field descriptor is generated.

Valid formats are "YYYY-MM" or "YYYYMM". When `x` is numeric or integer, it is assumed that it was a yearmonth in the format "YYYYMM" that was accidentally converted to numeric format.

**Value**

Will return an Date vector with `fielddescriptor` added as the 'fielddescriptor' attribute. The dates will be the first of the given month. Therefore, a 'yearmonth' "2024-05" is translated to a date "2024-05-01".

---

dp_write_data	<i>Write data of resource to file</i>
---------------	---------------------------------------

---

**Description**

Write data of resource to file

**Usage**

```
dp_write_data(x, ..., write_categories = TRUE)
```

```
## S3 method for class 'datapackage'
```

```
dp_write_data(
  x,
  resource_name,
  data,
  writer = "guess",
  ...,
  write_categories = TRUE
)
```

```
## S3 method for class 'dataresource'
```

```
dp_write_data(
  x,
  data,
  datapackage = dp_get_datapackage(x),
  writer = "guess",
  ...,
  write_categories = TRUE
)
```

**Arguments**

`x` the Data Package or Data Resource to which the data is to be written to.  
`...` additional arguments are passed on to the writer function.

<code>write_categories</code>	write both the data set <code>x</code> itself and any <code>categories</code> lists of fields in the data set.
<code>resource_name</code>	name of the Data Resource in the Data Package to which the data needs to be written.
<code>data</code>	<code>data.frame</code> with the data to write.
<code>writer</code>	the writer to use to write the data. This should be either a function accepting the Data Package, name of the Data Resource, the data and the <code>write_categories</code> argument or the character string "guess".
<code>datapackage</code>	the Data Package to which the data needs to be written.

### Details

When `writer = "guess"` the function will try to guess which writer to use based on the `format` and `mediatype` of the Data Resource.

### Value

The function doesn't return anything. It is called for its side effect of creating files in the directory of the Data Package.

---

<code>fwf_reader</code>	<i>Read the FWF-data for a Data Resource</i>
-------------------------	--

---

### Description

Read the FWF-data for a Data Resource

### Usage

```
fwf_reader(path, resource, convert_categories = c("no", "to_factor"), ...)
```

### Arguments

<code>path</code>	path to the data set.
<code>resource</code>	a Data Resource.
<code>convert_categories</code>	how to handle columns for which the field descriptor has a <code>categories</code> property. Passed on to <a href="#">dp_apply_schema</a> .
<code>...</code>	additional arguments are passed on to <a href="#">dp_apply_schema</a> .

### Value

Returns a `data.frame` with the data.

### See Also

Generally used by calling [dp\\_get\\_data](#).

---

new\_contributor      *Creating and Adding Contributors to a Data Package*

---

## Description

Creating and Adding Contributors to a Data Package

## Usage

```
new_contributor(  
  title,  
  role = c("contributor", "author", "publisher", "maintainer", "wrangler"),  
  path = NULL,  
  email = NULL,  
  organisation = NULL  
)  
  
dp_add_contributor(x, contributor)  
  
dp_add_contributor(x) <- value
```

## Arguments

title	A length 1 character vector with the full nam of the contributor.
role	The role of the contributor
path	A URL to e.g. a home page of the contributor
email	The email address of the contributor
organisation	The orgination the contributor belongs to.
x	The Data Package to which the contributor has to be added.
contributor	a contributor object
value	a contributor object

## Value

new\_contributor returns a list with the given properties. This function is meant to assist in creating valid contributors.

## Examples

```
dp <- open_datapackage(system.file(package = "datapackage", "examples/iris"))  
dp_contributors(dp)  
dp_contributors(dp) <- list(  
  new_contributor("John Doe", email = "j.doe@somewhere.org"),  
  list(title = "Jane Doe", role = "maintainer")  
)  
dp_add_contributor(dp) <- new_contributor("Janet Doe")
```

---

new_datapackage	<i>Create a new Data Package</i>
-----------------	----------------------------------

---

## Description

Create a new Data Package

## Usage

```
new_datapackage(path, name = NULL, title = NULL, description = NULL, ...)
```

## Arguments

path	The directory which will contain the Data Package or the filename in which to write the Data Package.
name	The name of the Data Package.
title	The title of the Data Package.
description	The description of the Data Package.
...	Ignored for now.

## Value

The directory of path, or the directory containing path if path is a file name, is created and the file with the Data Package information is created. When path is a directory a file datapackage.json is created. The function returns an editable datapackage object.

## Examples

```
dir <- tempdir()
dp <- new_datapackage(dir, name = "test-package")

dp_title(dp) <- "A Test Data Package"
dp_add_contributor(dp) <- new_contributor(title = "John Doe")
```

---

new\_dataresource      *Create a new Data Resource*

---

## Description

Create a new Data Resource

## Usage

```
new_dataresource(  
    name,  
    title = NULL,  
    description = NULL,  
    path = NULL,  
    format = NULL,  
    mediatype = NULL,  
    encoding = NULL,  
    bytes = NULL,  
    hash = NULL,  
    ...  
)
```

## Arguments

name	The name of the Data Resource.
title	The title of the Data Resource.
description	The description of the Data Resource.
path	the path of the Data Resource
format	the format of the Data Resource
mediatype	the mediatype of the Data Resource
encoding	the encoding of the Data Resource
bytes	the number of bytes of the Data Resource
hash	the hash of the Data Resource
...	additional arguments are added as additional properties. It is checked if these are valid.

## Value

Returns a dataresource object which is a list with the properties of the Data Resource.

## Examples

```
dir <- tempdir()
dp <- new_datapackage(dir, name = "test-package")

res <- new_dataresource(name = "iris")
dp_title(res) <- "The Iris Data Set"
dp_encoding(res) <- "UTF-8"
dp_mediatype(res) <- "text/csv"

# resource adds a resource if it doesn't yet exist or updates
# an existing resource
dp_resource(dp, "iris") <- res
```

---

open_datapackage	<i>Open a data package</i>
------------------	----------------------------

---

## Description

Open a data package

## Usage

```
open_datapackage(path, readonly = TRUE)
```

## Arguments

path	The filename or the data package description or the directory in which the data package is located.
readonly	Open the data package as a read-only data package or not. See 'details'

## Details

When path is a directory name, the function looks for the files 'datapackage.json' or 'datapackage.yaml' in the directory. Otherwise, the function assumes the file contains the description of the data package.

When the data package is read with `readonly = FALSE`, any operations reading properties from the data package read those properties directly from the file on disk. And setting the properties will change the file on disk. This ensures the file is always consistent.

## Value

Returns a list with the contents of the data package when `readonly = TRUE`. Otherwise an empty list is returned. In both cases the filename of the data package description (typically 'datapackage.json') and the director in which the data package is located are stored in attributes of the result.

---

PropertiesDatapackage *Getting and setting properties of Data Packages*

---

**Description**

Getting and setting properties of Data Packages

**Usage**

```
dp_contributors(x, ...)

dp_contributors(x) <- value

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

## S3 replacement method for class 'datapackage'
dp_contributors(x) <- value

dp_name(x)

## S3 method for class 'datapackage'
dp_name(x)

dp_name(x) <- value

## S3 replacement method for class 'datapackage'
dp_name(x) <- value

dp_title(x)

## S3 method for class 'datapackage'
dp_title(x)

dp_title(x) <- value

## S3 replacement method for class 'datapackage'
dp_title(x) <- value

dp_description(x, ..., first_paragraph = FALSE, dots = FALSE)

## S3 method for class 'datapackage'
dp_description(x, ..., first_paragraph = FALSE, dots = FALSE)

dp_description(x) <- value

## S3 replacement method for class 'datapackage'
```

```
dp_description(x) <- value

dp_keywords(x, ...)

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

dp_keywords(x) <- value

## S3 replacement method for class 'datapackage'
dp_keywords(x) <- value

dp_created(x, ...)

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

dp_created(x) <- value

## S3 replacement method for class 'datapackage'
dp_created(x) <- value

dp_id(x, ...)

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

dp_id(x) <- value

## S3 replacement method for class 'datapackage'
dp_id(x) <- value
```

### Arguments

x	a datapackage object.
...	used to pass additional arguments to other methods.
value	the new value of the property.
first_paragraph	Only return the first paragraph of the description.
dots	When returning only the first paragraph indicate missing paragraphs with ...

### Value

Either returns the property or modifies the object.

### See Also

See [dp\\_resource](#) for methods for getting and setting the resources of a Data Package.

See [PropertiesDataresource](#) and [PropertiesFielddescriptor](#) for methods for Data Resources and Field Descriptors respectively. Also see [dp\\_property](#) for a generic method for getting and setting properties. These functions can also be used to get and set 'unofficial' properties'

---

PropertiesDataresource

*Getting and setting properties of Data Resources*

---

## Description

Getting and setting properties of Data Resources

## Usage

```
## S3 method for class 'dataresource'  
dp_name(x)  
  
## S3 replacement method for class 'dataresource'  
dp_name(x) <- value  
  
## S3 method for class 'dataresource'  
dp_title(x)  
  
## S3 replacement method for class 'dataresource'  
dp_title(x) <- value  
  
## S3 method for class 'dataresource'  
dp_description(x, ..., first_paragraph = FALSE, dots = FALSE)  
  
## S3 replacement method for class 'dataresource'  
dp_description(x) <- value  
  
dp_path(x, ...)  
  
dp_path(x) <- value  
  
## S3 method for class 'dataresource'  
dp_path(x, full_path = FALSE, ...)  
  
## S3 replacement method for class 'dataresource'  
dp_path(x) <- value  
  
dp_format(x, ...)  
  
dp_format(x) <- value  
  
## S3 method for class 'dataresource'
```

```
dp_format(x, default = FALSE, ...)  
  
## S3 replacement method for class 'dataresource'  
dp_format(x) <- value  
  
dp_mediatype(x, ...)  
  
dp_mediatype(x) <- value  
  
## S3 method for class 'dataresource'  
dp_mediatype(x, ...)  
  
## S3 replacement method for class 'dataresource'  
dp_mediatype(x) <- value  
  
dp_encoding(x, default = FALSE, ...)  
  
dp_encoding(x) <- value  
  
## S3 method for class 'dataresource'  
dp_encoding(x, default = FALSE, ...)  
  
## S3 replacement method for class 'dataresource'  
dp_encoding(x) <- value  
  
dp_bytes(x, ...)  
  
dp_bytes(x) <- value  
  
## S3 method for class 'dataresource'  
dp_bytes(x, ...)  
  
## S3 replacement method for class 'dataresource'  
dp_bytes(x) <- value  
  
dp_hash(x, ...)  
  
dp_hash(x) <- value  
  
## S3 method for class 'dataresource'  
dp_hash(x, ...)  
  
## S3 replacement method for class 'dataresource'  
dp_hash(x) <- value  
  
## S3 replacement method for class 'fielddescriptor'  
dp_name(x) <- value
```

```

## S3 replacement method for class 'fielddescriptor'
dp_title(x) <- value

## S3 method for class 'fielddescriptor'
dp_description(x, ..., first_paragraph = FALSE, dots = FALSE)

## S3 replacement method for class 'fielddescriptor'
dp_format(x) <- value

## S3 replacement method for class 'fielddescriptor'
dp_type(x) <- value

dp_schema(x)

## S3 method for class 'dataresource'
dp_schema(x)

```

### Arguments

x	a dataresource object.
value	the new value of the property.
...	used to pass additional arguments to other methods.
first_paragraph	Only return the first paragraph of the description.
dots	When returning only the first paragraph indicate missing paragraphs with ...
full_path	Return the full path including the path to the Data Package and not only the path relative to the Data Package. This is only relevant for relative paths.
default	return the default value if the property had a default value and the property is not set.

### Value

Either returns the property or modifies the object. If the property of not set NULL is returned (unless default = TRUE).

### See Also

See [PropertiesDatapackage](#) and [PropertiesFielddescriptor](#) for methods for Data Packages and Field Descriptors respectively. Also see [dp\\_property](#) for a generic method for getting and setting properties. These functions can also be used to get and set 'unofficial' properties'

---

PropertiesFielddescriptor

*Getting and setting properties of Data Resources*

---

## Description

Getting and setting properties of Data Resources

## Usage

```
## S3 method for class 'fielddescriptor'
dp_name(x)

## S3 method for class 'fielddescriptor'
dp_title(x)

## S3 replacement method for class 'fielddescriptor'
dp_description(x) <- value

## S3 method for class 'fielddescriptor'
dp_format(x, ...)

dp_type(x)

dp_type(x) <- value

## S3 method for class 'fielddescriptor'
dp_type(x)
```

## Arguments

x	a fielddescriptor object.
value	the new value of the property.
...	used to pass additional arguments to other methods.

## Value

Either returns the property or modifies the object. If the property is not set NULL is returned (unless default = TRUE).

## See Also

See [PropertiesDatapackage](#) and [PropertiesDataresource](#) for methods for Data Packages and Data Resources respectively. Also see [dp\\_property](#) for a generic method for getting and setting properties. These functions can also be used to get and set 'unofficial' properties'

# Index

as.codelist, 23

code, 23

csv\_reader, 3

csv\_writer, 4

dp\_add\_contributor (new\_contributor), 32

dp\_add\_contributor<- (new\_contributor), 32

dp\_add\_reader, 4, 15

dp\_add\_writer, 5

dp\_apply\_schema, 3, 6, 31

dp\_bytes (PropertiesDataresource), 38

dp\_bytes<- (PropertiesDataresource), 38

dp\_categorieslist, 7

dp\_check\_dataresource, 8, 9

dp\_check\_field, 9, 9

dp\_contributors  
(PropertiesDatapackage), 36

dp\_contributors<-  
(PropertiesDatapackage), 36

dp\_created (PropertiesDatapackage), 36

dp\_created<- (PropertiesDatapackage), 36

dp\_description (PropertiesDatapackage), 36

dp\_description.dataresource  
(PropertiesDataresource), 38

dp\_description.fielddescriptor  
(PropertiesDataresource), 38

dp\_description<-  
(PropertiesDatapackage), 36

dp\_description<- .dataresource  
(PropertiesDataresource), 38

dp\_description<- .fielddescriptor  
(PropertiesFielddescriptor), 41

dp\_encoding, 19

dp\_encoding (PropertiesDataresource), 38

dp\_encoding<- (PropertiesDataresource), 38

dp\_field, 10

dp\_field\_names, 10

dp\_format (PropertiesDataresource), 38

dp\_format.fielddescriptor  
(PropertiesFielddescriptor), 41

dp\_format<- (PropertiesDataresource), 38

dp\_generate\_dataresource, 11, 22

dp\_generate\_fielddescriptor, 11, 12, 21

dp\_get\_connection, 14

dp\_get\_data, 3, 5, 14, 15, 16, 17, 31

dp\_get\_datapackage, 16

dp\_hash (PropertiesDataresource), 38

dp\_hash<- (PropertiesDataresource), 38

dp\_id (PropertiesDatapackage), 36

dp\_id<- (PropertiesDatapackage), 36

dp\_keywords (PropertiesDatapackage), 36

dp\_keywords<- (PropertiesDatapackage), 36

dp\_load\_from\_datapackage, 16

dp\_mediatype (PropertiesDataresource), 38

dp\_mediatype<-  
(PropertiesDataresource), 38

dp\_name, 19

dp\_name (PropertiesDatapackage), 36

dp\_name.dataresource  
(PropertiesDataresource), 38

dp\_name.fielddescriptor  
(PropertiesFielddescriptor), 41

dp\_name<- (PropertiesDatapackage), 36

dp\_name<- .dataresource  
(PropertiesDataresource), 38

dp\_name<- .fielddescriptor  
(PropertiesDataresource), 38

dp\_nresources, 17

dp\_path (PropertiesDataresource), 38

dp\_path<- (PropertiesDataresource), 38

dp\_properties, 17

dp\_property, 18, 18, 38, 40, 41

dp\_property<- (dp\_property), 18

dp\_resource, [19](#), [37](#)  
dp\_resource<- (dp\_resource), [19](#)  
dp\_resource\_names, [21](#)  
dp\_resources<-, [20](#)  
dp\_save\_as\_datapackage, [21](#)  
dp\_schema (PropertiesDataresource), [38](#)  
dp\_title (PropertiesDatapackage), [36](#)  
dp\_title.dataresource  
    (PropertiesDataresource), [38](#)  
dp\_title.fielddescriptor  
    (PropertiesFielddescriptor), [41](#)  
dp\_title<- (PropertiesDatapackage), [36](#)  
dp\_title<-.dataresource  
    (PropertiesDataresource), [38](#)  
dp\_title<-.fielddescriptor  
    (PropertiesDataresource), [38](#)  
dp\_to\_boolean, [7](#), [22](#)  
dp\_to\_code, [6](#), [23](#), [25](#)  
dp\_to\_date, [7](#), [24](#)  
dp\_to\_datetime, [7](#), [24](#)  
dp\_to\_factor, [6](#), [23](#), [25](#)  
dp\_to\_integer, [7](#), [26](#)  
dp\_to\_number, [6](#), [7](#), [27](#)  
dp\_to\_string, [7](#), [27](#)  
dp\_to\_time, [28](#)  
dp\_to\_year, [29](#)  
dp\_to\_yearmonth, [7](#), [29](#)  
dp\_type (PropertiesFielddescriptor), [41](#)  
dp\_type<- (PropertiesFielddescriptor),  
    [41](#)  
dp\_type<-.fielddescriptor  
    (PropertiesDataresource), [38](#)  
dp\_write\_data, [22](#), [30](#)  
  
fread, [3](#)  
fwf\_reader, [31](#)  
  
iso8601totime, [28](#)  
isTRUE, [9](#)  
  
new\_contributor, [32](#)  
new\_datapackage, [22](#), [33](#)  
new\_dataresource, [34](#)  
  
open\_datapackage, [17](#), [35](#)  
  
PropertiesDatapackage, [36](#), [40](#), [41](#)  
PropertiesDataresource, [38](#), [38](#), [41](#)  
PropertiesFielddescriptor, [38](#), [40](#), [41](#)  
  
read.csv, [3](#)