Package 'csvwr'

June 14, 2021

Type Package

Title Read and Write CSV on the Web (CSVW) Tables and Metadata

Version 0.1.5

Author Robin Gower

Maintainer Robin Gower <csvwr@infonomics.ltd.uk>

Description Provide functions for reading and writing CSVW - i.e. CSV tables and JSON metadata. The metadata helps interpret CSV by setting the types and variable names.

License GPL-3

URL https://robsteranium.github.io/csvwr/, https://github.com/Robsteranium/csvwr

BugReports https://github.com/Robsteranium/csvwr/issues

Encoding UTF-8

Suggests testthat (>= 3.0.0), knitr, markdown, rmarkdown

Imports magrittr, jsonlite, purrr, readr, stringr, rlang

Config/testthat/edition 3

RoxygenNote 7.1.1

VignetteBuilder knitr, rmarkdown

NeedsCompilation no

Repository CRAN

Date/Publication 2021-06-14 07:30:15 UTC

R topics documented:

add_dataframe *Add data frame to csvw table annotation*

Description

Add data frame to csvw table annotation

Usage

```
add_dataframe(table, filename, dialect, group_schema)
```

Arguments

table	a csvw:Table annotation
filename	a filename/ URL for the csv table
dialect	a list describing the dialect for reading the csv file
group_schema	a fallback tableSchema from the table group

Value

a table annotation with a dataframe attribute added with data frame holding the contents of the table

base_uri *Retrieve the base URI from configuration*

Description

Retrieve the base URI from configuration

Usage

```
base_uri()
```

Value

returns the value of csvwr_base_uri option, defaulting to example.net

Examples

```
## Not run:
base_uri() # returns default

options(csvwr_base_uri="http://www.w3.org/2013/csvw/tests/")
base_uri()

## End(Not run)
```

base_url *Determine the base URL for CSVW metadata*

Description

Determine the base URL for CSVW metadata

Usage

```
base_url(metadata, location)
```

Arguments

metadata	the csvw metadata
location	where the metadata was originally located

Value

A string containing the base URL

coalesce_truth *Coalesce value to truthiness*

Description

Determine whether the input is true, with missing values being interpreted as false.

Usage

```
coalesce_truth(x)
```

Arguments

x	logical, NA or NULL

Value

FALSE if x is anything but TRUE

compact_json_ld *Compact objects to values*

Description

Follows the rules for JSON-LD to JSON conversion set out in the csv2json standard.

Usage

```
compact_json_ld(value)
```

Arguments

value	an element from a list (could be a vector or another list)

Value

A compacted value.

create_metadata *Create tabular metadata from a list of tables*

Description

The table annotations should each be a list with keys for `url` and `tableSchema`. You can use `derive_table_schema` to derive a schema from a data frame.

Usage

```
create_metadata(tables)
```

Arguments

tables	a list of `csvw:table` annotations

Value

a list describing a tabular metadata annotation

Examples

```
d <- data.frame(foo="bar")
table <- list(url="filename.csv", tableSchema=derive_table_schema(d))
create_metadata(tables=list(table))
```

csvwr *csvwr: Read and write CSV on the Web (CSVW)*

Description

Read and write csv tables annotated with csvw metadata. This helps to ensure consistent processing and reduce the amount of manual work needed to parse and prepare data before it can be used in analysis.

Getting started

The best place to start is the Reading and Writing CSVW vignette.

Reading annotated tables

- `read_csvw` Parse a table group
- `read_csvw_dataframe` Parse a table group and extract the first data frame

Writing table annotations

- `derive_table_schema` Derive table schema from a data frame
- `create_metadata` Create a table group annotation
- `derive_metadata` Derive an annotation from a csv file

csvwr_example *Get path to csvwr example*

Description

The csvwr package includes some example csvw files in it's `inst/extdata` directory. You can use this function to find them.

Usage

```
csvwr_example(path = NULL)
```

Arguments

`path`	The filename. If `NULL`, the example files will be listed.

Details

Inspired by `readr::readr_example()`

Value

either a file path or a vector of filenames

Examples

```
csvwr_example()
csvwr_example("computer-scientists.csv")
```

---

csvw_to_list *Convert a csvw metadata to a list (csv2json)*

Description

Convert a csvw metadata to a list (csv2json)

Usage

```
csvw_to_list(csvw)
```

Arguments

csvw	a csvw metadata list

Value

a list following the csv2json translation rules

Examples

```
## Not run:
csvw_to_list(read_csvw("example.csv"))

## End(Not run)
```

---

datatype_to_type *Map csvw datatypes to R types*

Description

Translate csvw datatypes to R types. This implementation currently targets readr::cols column specifications.

Usage

```
datatype_to_type(datatypes)
```

Arguments

datatypes	a list of csvw datatypes

Value

a readr::cols specification - a list of collectors

Examples

```
## Not run:
cspec <- datatype_to_type(list("double", list(base="date", format="yyyy-MM-dd")))
readr::read_csv(readr::readr_example("challenge.csv"), col_types=cspec)

## End(Not run)
```

---

default_dialect *CSVW default dialect*

Description

The CSVW Default Dialect specification described in CSV Dialect Description Format.

Usage

```
default_dialect
```

Format

An object of class list of length 13.

Value

a list specifying a default csv dialect

---

default_schema *Create a default table schema given a csv file and dialect*

Description

If neither the table nor the group have a tableSchema annotation, then this default schema will used.

Usage

```
default_schema(filename, dialect)
```

Arguments

filename	a csv file
dialect	specification of the csv's dialect

Value

a table schema

---

derive_metadata *Derive csvw metadata from a csv file*

Description

Derive csvw metadata from a csv file

Usage

```
derive_metadata(filename)
```

Arguments

filename	a csv file

Value

a list of csvw metadata

Examples

```
derive_metadata(csvwr_example("computer-scientists.csv"))
```

---

derive_table_schema *Derive csvw table schema from a data frame*

Description

Derive csvw table schema from a data frame

Usage

```
derive_table_schema(d)
```

Arguments

d	a data frame

Value

a list describing a csvw:tableSchema

Examples

```
derive_table_schema(data.frame(a=1,b=2))
```

---

extract_table *Extract a referenced table from CSVW metadata*

Description

Extract a referenced table from CSVW metadata

Usage

```
extract_table(csvw, reference)
```

Arguments

csvw	the metadata
reference	a foreign key reference expressed as a list containing either a reference attribute or a schemaReference attribute

Value

a csvw table

---

find_existing_file *Find the first existing file from a set of candidates*

Description

Find the first existing file from a set of candidates

Usage

```
find_existing_file(filenames)
```

Arguments

filenames	a vector of candidates

Value

If one of the filenames passed is found, then the first is returned. If none of the filenames exist, NULL is returned

find_metadata *Find metadata for a tabular file*

Description

Searches through the default locations attempting to locate metadata.

Usage

```
find_metadata(filename)
```

Arguments

filename	a csv file

Value

a uri for the metadata, or null if none were found

is_absolute_url *Does the string provide an absolute URL*

Description

Does the string provide an absolute URL

Usage

```
is_absolute_url(string)
```

Arguments

string	the url, path or template

Value

true if the string is an absolute url

is_non_core_annotation *Determine if an annotation is non-core*

Description

Checks if the annotation is non-core, and should thus be treated as a json-ld note.

Usage

```
is_non_core_annotation(property)
```

Arguments

property	a list element

Value

TRUE the annotation is core, FALSE otherwise

json_ld_to_json *Convert json-ld annotation to json*

Description

Follows the rules for JSON-LD to JSON conversion set out in the csv2json standard.

Usage

```
json_ld_to_json(property)
```

Arguments

property	a json-ld annotation (single list element)

Value

A compacted list element

list_of_lists_to_df *Parse list of lists specification into a data frame*

Description

Parse list of lists specification into a data frame

Usage

```
list_of_lists_to_df(ll)
```

Arguments

ll	a list of lists

Value

a data frame with a row per list

locate_metadata *Locate metadata for a table*

Description

Follows the procedure defined in the csvw model:

Usage

```
locate_metadata(filename, metadata)
```

Arguments

filename	a path for a csv table or a json metadata document
metadata	optional user metadata

Details

1. Metadata supplied by the user
2. Metadata referenced by a link header
3. Metadata located through default paths
4. Metadata embedded in the file

We extend this to use the derive_metadata function to inspect the table itself.

Value

csvw metadata list

locate_table *Locate csv data table*

Description

Locate csv data table

Usage

```
locate_table(filename, url)
```

Arguments

filename	the file passed to read_csvw in the first place (could be the csv or json annotations)
url	the location of the the table as defined in the metadata

Value

The location of the table

location_configuration *Identify metadata location configurations for a tabular file*

Description

Returns default locations. Will ultimately retrieve remote configuration

Usage

```
location_configuration(filename)
```

Arguments

filename	a csv file

Value

a character vector of URI templates

normalise_metadata *Normalise metadata*

Description

The spec defines a normalisation process.

Usage

```
normalise_metadata(metadata, location)
```

Arguments

metadata	a csvw metadata list
location	the location of the metadata

Value

metadata with normalised properties

normalise_property *Normalise an annotation property*

Description

This follows the normalisation process set out in the csvw specification.

Usage

```
normalise_property(property, base_url)
```

Arguments

property	an annotation property (a list)
base_url	the base URL for normalisation

Value

a property (list) a

normalise_url *Normalise a URL*

Description

Ensures that a url is specified absolutely with reference to a base

Usage

```
normalise_url(url, base)
```

Arguments

url	a string
base	the base to use for normalisation

Value

A string containing a normalised URL

parse_columns *Parse columns schema*

Description

Parse columns schema

Usage

```
parse_columns(columns)
```

Arguments

columns	a list of lists specification of columns

Value

a data frame with a row per column specification

parse_metadata *Parse metadata*

Description

Coerces the metadata to ensure it describes a table group. Retrieves any linked tableSchema.

Usage

```
parse_metadata(metadata, location)
```

Arguments

metadata	a csvw metadata list
location	the location of the metadata

Value

metadata coerced into a table group description

read_csvw *Read CSV on the Web*

Description

If the argument to filename is a json metadata document, this will be used to find csv files for each table using the value of csvw:url.

Usage

```
read_csvw(filename, metadata = NULL)
```

Arguments

filename	a path for a csv table or a json metadata document
metadata	optional user metadata

Details

If the argument to filename is a csv file, and no metadata is provided, an attempt is made to derive metadata.

If the argument to filename is a csv file, and the metadata is provided, then the given csv will override the value of csvw:url.

The csvw metadata is returned as a list. In each table in the table group, an element named dataframe is added which provides the contents of the csv table parsed into a data frame using the table schema.

Value

csvw metadata list, with a `dataframe` property added to each table

Examples

```
## Not run:
read_csvw("metadata.json")
read_csvw("table.csv", "metadata.json")

## End(Not run)
```

`read_csvw_dataframe` *Read a data frame from the first table in a csvw*

Description

Wrapper around `read_csvw` convenient when you're only interested in the data and there's only one table

Usage

```
read_csvw_dataframe(filename, metadata = NULL)
```

Arguments

`filename`	a path for a csv table or a json metadata document
`metadata`	optional user metadata

Value

A data frame parsed using the table schema

`read_metadata` *Read and parse CSVW Metadata*

Description

Reads in a json document as a list, transforming columns specifications into a dataframe.

Usage

```
read_metadata(filename)
```

Arguments

`filename`	a path for a json metadata document

Value

csvw metadata list

`render_cell` *Serialise cell values for JSON representation*

Description

Serialise cell values for JSON representation

Usage

```
render_cell(cell)
```

Arguments

`cell`	a typed value

Value

a representation comparable with the JSON representation (typically a string)

`render_uri_templates` *Render URI templates*

Description

Interpolate variable bindings into a URI template.

Usage

```
render_uri_templates(templates, bindings = NULL, ...)
```

Arguments

`templates`	a character vector with URI templates
`bindings`	a list of variable bindings to be interpolated into templates
`...`	further bindings specified as named function arguments

Details

This doesn't yet implement the whole of RFC 6570, just enough to make the tests pass

You can bind variables by passing a list to the explicit `bindings` argument, or variadically with `...` by naming arguments according to the variable name you wish to bind.

Value

a character vector with the expanded URI

Examples

```
render_uri_templates("{+url}/resource?query=value", list(url="http://example.net"))
render_uri_templates("{+url}", url="http://example.net")
```

`resolve_url` *Resolve one URL against another*

Description

Resolve one URL against another

Usage

```
resolve_url(url1, url2)
```

Arguments

`url1`	the base url
`url2`	a relative url

Value

A single absolute url

`rlmap` *Recursive lmap*

Description

Applies function `.f` to each list-element in `.x` as per `purrr::lmap`. If the value of the list-element is itself a list, then the function is applied to that in turn. The process is followed recursively until an atomic value at the leaf nodes of the list is found. If `.f` modifies the name, it is thrown away and replaced by the original name.

Usage

```
rlmap(.x, .f, ...)
```

Arguments

.x	a list
.f	a function (called with elements of .x as the first argument)
...	further arguments passed to the function .f

Value

A list

---

rmap *Recursive map*

Description

Applies function .f to each element in .x as per purrr::map. If the value of the element is itself a list, then the function is applied to that in turn. The process is followed recursively until an atomic value at the leaf nodes of the list is found.

Usage

```
rmap(.x, .f)
```

Arguments

.x	a list
.f	a function (called with elements of .x as the first argument)

Value

A list

---

set_uri_base *Set the base of a URI template*

Description

Set the base of a URI template

Usage

```
set_uri_base(t, url)
```

Arguments

t	a character vector of URI templates
url	a filename url being used as a context (string)

Value

a character vector of templates with base paths/ domains set appropriately

---

`table_to_list` *Convert a table to a list*

Description

Follows the pattern for csv2json

Usage

```
table_to_list(table, group_schema, dialect)
```

Arguments

`table`	the csvw table
`group_schema`	a default schema
`dialect`	the dialect for reading the table from the csv file

Value

a list representation of the table's contents

`transform_datetime_format` *Transform date/time format string from Unicode TR35 to POSIX 1003.1*

Description

As per the csvw specification for date and time formats we accept format strings using the date field symbols defined in unicode TR35. These are converted to POSIX 1003.1 date format strings for use in `base::strptime()` or `readr::parse_date()`/`readr::parse_datetime()`.

Usage

```
transform_datetime_format(format_string)
```

Arguments

`format_string`	a UAX35 date format string

Value

a POSIX date format string

Examples

```
## Not run:
fmt <- transform_datetime_format("dd.MM.yyyy")
strptime("01.01.2001", format=fmt)

## End(Not run)
```

---

try_add_dataframe *Try to add a dataframe to the table*

Description

If this fails, a list describing the error is added instead

Usage

```
try_add_dataframe(table, ...)
```

Arguments

table	a csvw:Table annotation
...	arguments to add_dataframe

Value

A table annotation with a dataframe attribute added with data frame holding the contents of the table or an error.

---

type_to_datatype *Map R types to csvw datatype*

Description

Translate R types to csvw datatypes. Acts as an inverse of datatype_to_type but doesn't provide a 1:1 correspondence.

Usage

```
type_to_datatype(types)
```

Arguments

types	a list of R types

Value

a list of csvw datatypes

unlist1 *Unlist unless the list-elements are themselves lists*

Description

Convert a list of elements to a vector. Unlike `base::unlist` this doesn't convert the elements of inner lists to vector elements. Thus only a list a single layer deep is flattened to a vector.

Usage

```
unlist1(l)
```

Arguments

`l`	a list

Value

A list of lists or a vector

validate_csvw *Validate CSVW specification*

Description

Follows the csvw table validation procedure.

Usage

```
validate_csvw(csvw)
```

Arguments

`csvw`	a csvw metadata specification (a list)

Value

a validation report (list)

validate_referential_integrity *Validate the referential integrity of a csvw table group*

Description

Fails if foreign keys aren't found in the referenced tables

Usage

```
validate_referential_integrity(csvw)
```

Arguments

csvw	the metadata annotation

Value

a list specifying any foreign key violations

vec_depth *Calculate depth of vector safely*

Description

Like `purrr::vec_depth` but doesn't attempt to descend into errors

Usage

```
vec_depth(x)
```

Arguments

x	a vector

Value

An integer

Index