Package 'fetch'

February 11, 2024

Type Package

Title Fetch Data from Various Data Sources

Version 0.1.5

Maintainer David Bosak <dbosak01@gmail.com>

Description Contains functions to fetch data from various data sources. The user first creates a catalog of objects from a data source, then fetches data from the catalog. The package provides an easy way to access data from many different types of sources.

Encoding UTF-8

License CC0

URL https://fetch.r-sassy.org

BugReports https://github.com/dbosak01/fetch/issues

Depends R (>= 4.1), common

Imports readr, readxl, haven, crayon, tibble, foreign

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

RoxygenNote 7.3.1

NeedsCompilation no

Author David Bosak [aut, cre],
Kevin Kramer [ctb],
Archytas Clinical Solutions [cph]

Repository CRAN

Date/Publication 2024-02-11 00:30:02 UTC

R topics documented:

`catalog` *Create a data source catalog*

Description

The `catalog` function returns a data catalog for a data source. A data catalog is like a collection of data dictionaries for all the datasets in the data source. The catalog allows you to examine the datasets in the data source without yet loading anything into memory. Once you decide which data items you want to load, use the `fetch` function to load that item into memory.

Usage

```
catalog(source, engine, pattern = NULL, where = NULL, import_specs = NULL)
```

Arguments

`source`	The source for the data. This parameter is required. Normally the source is passed as a full or relative path.
`engine`	The data engine to use for this data source. This parameter is required. The available data engines are available on the `engines` enumeration. For example, `engines$csv` will specify the CSV engine, and `engines$rdata` will specify the RDATA engine.
`pattern`	A pattern to use when loading data items from the data source. The pattern can be a name or a vector of names. Names also accept wildcards. The supplied pattern will be used to filter which data items are loaded into the catalog. For example, the pattern `pattern = "AD*"` will load only datasets that start with "AD".
`where`	A where expression to use when fetching the data. This expression will apply to all fetch operations on this catalog. The where expression should be defined with the Base R `expression` function. The expression is unquoted and can use any Base R operators or functions.
`import_specs`	The import specs to use for any fetch operation on this catalog. The import spec can be used to control the data types on the incoming columns. You can create separate import specs for each dataset, or one import spec to use for all datasets. See the `import_spec` and `specs` functions for more information about this capability.

Value

The loaded data catalog, as class "dcat". The catalog will be a list of data dictionaries. Each data dictionary is a tibble.

See Also

The `fetch` function to retrieve data from the catalog, and the `import_spec` function to create import specifications.

Examples

```
# Get data directory
pkg <- system.file("extdata", package = "fetch")

# Create catalog
ct <- catalog(pkg, engines$csv)

# Example 1: Catalog all rows

# View catalog
ct
# data catalog: 6 items
# - Source: C:/packages/fetch/inst/extdata
# - Engine: csv
# - Items:
  # data item 'ADAE': 56 cols 150 rows
  # data item 'ADEX': 17 cols 348 rows
  # data item 'ADPR': 37 cols 552 rows
  # data item 'ADPSGA': 42 cols 695 rows
  # data item 'ADSL': 56 cols 87 rows
  # data item 'ADVS': 37 cols 3617 rows

# View catalog item
ct$ADEX
# data item 'ADEX': 17 cols 348 rows
# - Engine: csv
# - Size: 70.7 Kb
# - Last Modified: 2020-09-18 14:30:22
#    Name   Column     Class Label Format NAs MaxChar
# 1  ADEX  STUDYID character  <NA>     NA   0       3
# 2  ADEX  USUBJID character  <NA>     NA   0      10
# 3  ADEX   SUBJID character  <NA>     NA   0       3
# 4  ADEX   SITEID character  <NA>     NA   0       2
# 5  ADEX     TRTP character  <NA>     NA   8       5
# 6  ADEX    TRTPN   numeric  <NA>     NA   8       1
# 7  ADEX     TRTA character  <NA>     NA   8       5
# 8  ADEX    TRTAN   numeric  <NA>     NA   8       1
# 9  ADEX   RANDFL character  <NA>     NA   0       1
# 10 ADEX    SAFFL character  <NA>     NA   0       1
# 11 ADEX   MITTFL character  <NA>     NA   0       1
# 12 ADEX  PPROTFL character  <NA>     NA   0       1
# 13 ADEX    PARAM character  <NA>     NA   0      45
```

```
# 14 ADEX  PARAMCD character  <NA>     NA   0       8
# 15 ADEX   PARAMN   numeric  <NA>     NA   0       1
# 16 ADEX     AVAL   numeric  <NA>     NA  16       4
# 17 ADEX AVALCAT1 character  <NA>     NA  87      10


# Example 2: Catalog with where expression
ct <- catalog(pkg, engines$csv, where = expression(SUBJID == '049'))

# View catalog item - Now only 4 rows
ct$ADEX
# data item 'ADEX': 17 cols 4 rows
#- Where: SUBJID == "049"
#- Engine: csv
#- Size: 4.5 Kb
#- Last Modified: 2020-09-18 14:30:22
#Name   Column     Class Label Format NAs MaxChar
#1  ADEX  STUDYID character  <NA>     NA   0       3
#2  ADEX  USUBJID character  <NA>     NA   0      10
#3  ADEX   SUBJID character  <NA>     NA   0       3
#4  ADEX   SITEID character  <NA>     NA   0       2
#5  ADEX     TRTP character  <NA>     NA   0       5
#6  ADEX    TRTPN   numeric  <NA>     NA   0       1
#7  ADEX     TRTA character  <NA>     NA   0       5
#8  ADEX    TRTAN   numeric  <NA>     NA   0       1
#9  ADEX   RANDFL character  <NA>     NA   0       1
#10 ADEX    SAFFL character  <NA>     NA   0       1
#11 ADEX   MITTFL character  <NA>     NA   0       1
#12 ADEX  PPROTFL character  <NA>     NA   0       1
#13 ADEX    PARAM character  <NA>     NA   0      45
#14 ADEX  PARAMCD character  <NA>     NA   0       8
#15 ADEX   PARAMN   numeric  <NA>     NA   0       1
#16 ADEX     AVAL   numeric  <NA>     NA   0       4
#17 ADEX AVALCAT1 character  <NA>     NA   1      10
```

`engines` *A list of engine types*

Description

The engines enumeration contains all possible options for the "engine" parameter of the `catalog` function. Use this enumeration to specify what kind of data you would like to load. Options are: csv, dbf, rda, rds, rdata, sas7bdat, xls, xlsx, and xpt.

Usage

```
engines
```

Format

An object of class etype of length 9.

Value

The engine parameter string.

See Also

The engines enumeration is used on the catalog function. See that function documentation for additional details.

Examples

```
#' # Get data directory
pkg <- system.file("extdata", package = "fetch")

# Create catalog
ct <- catalog(pkg, engines$csv)

# Example 1: Catalog all rows

# View catalog
ct
# data catalog: 6 items
# - Source: C:/packages/fetch/inst/extdata
# - Engine: csv
# - Items:
  # data item 'ADAE': 56 cols 150 rows
  # data item 'ADEX': 17 cols 348 rows
  # data item 'ADPR': 37 cols 552 rows
  # data item 'ADPSGA': 42 cols 695 rows
  # data item 'ADSL': 56 cols 87 rows
  # data item 'ADVS': 37 cols 3617 rows
```

---

import_spec *Create an Import Specification*

Description

A function to create the import specifications for a particular data file. This information can be used on the catalog or fetch functions to correctly assign the data types for columns on imported data. The import specifications are defined as name/value pairs, where the name is the column name and the value is the data type indicator. Available data type indicators are 'guess', 'logical', 'character', 'integer', 'numeric', 'date', 'datetime', and 'time'.

Also note that multiple import specifications can be combined into a collection, and assigned to an entire catalog. See the specs function for an example of using a specs collection.

Usage

```
import_spec(..., na = NULL, trim_ws = NULL)
```

Arguments

`...`	Named pairs of column names and column data types, separated by commas. Available types are: 'guess', 'logical', 'character', 'integer', 'numeric', 'date', 'datetime', and 'time'. The date/time data types accept an optional input format. To supply the input format, append it after the data type following an equals sign, e.g.: 'date=%d%b%Y' or 'datetime=%d-%m-%Y %H:%M:%S'. Default is NULL, meaning no column types are specified, and the function should make its best guess for each column.
`na`	A vector of values to be treated as NA. For example, the vector `c('', ' ')` will cause empty strings and single blanks to be converted to NA values. Default is NULL, meaning the value of the `na` parameter will be taken from the `specs` function. Any value supplied on the `import_spec` function will override the value from the `specs` function.
`trim_ws`	Whether or not to trim white space from the input data values. The default is NULL, meaning the value of the `trim_ws` parameter will be taken from the `specs` function. Any value supplied on the `import_spec` function will override the value from the `specs` function.

Value

The import specification object. The class of the object will be "import_spec".

Date/Time Format Codes

Below are some common date formatting codes. For a complete list, see the documentation for the `strptime` function:

- %d = day as a number
- %a = abbreviated weekday
- %A = unabbreviated weekday
- %m = month number
- %b = abbreviated month name
- %B = unabbreviated month name
- %y = 2-digit year
- %Y = 4-digit year
- %H = hour
- %M = minute
- %S = second
- %p = AM/PM indicator

See Also

fetch to retrieve data, and specs for creating a collection of import specs.

Other specs: print.specs(), read.specs(), specs(), write.specs()

Examples

```
# Get sample data directory
pkg <- system.file("extdata", package = "fetch")

# Create import spec
spc <- import_spec(TRTSDT = "date=%d%b%Y",
                   TRTEDT = "date=%d%b%Y")

# Create catalog without filter
ct <- catalog(pkg, engines$csv, import_specs = spc)

# Get dictionary for ADVS with Import Spec
d <- ct$ADVS

# Observe data types for TRTSDT and TRTEDT are now Dates
d[d$Column %in% c("TRTSDT", "TRTEDT"), ]
# data item 'ADVS': 37 cols 3617 rows
#- Engine: csv
#- Size: 1.1 Mb
#- Last Modified: 2020-09-18 14:30:22
#   Name Column Class Label Format NAs MaxChar
#16 ADVS TRTSDT  Date  <NA>     NA  54      10
#17 ADVS TRTEDT  Date  <NA>     NA 119      10
```

---

print.dcat *Print a data catalog*

Description

A class-specific instance of the print function for a data catalog. The function prints the catalog in a summary manner. Use verbose = TRUE option to print the catalog as a list.

Usage

```
## S3 method for class 'dcat'
print(x, ..., verbose = FALSE)
```

Arguments

x	The catalog to print.
...	Any follow-on parameters.
verbose	Whether or not to print the catalog in verbose style. By default, the parameter is FALSE, meaning to print in summary style.

Value

The object, invisibly.

Examples

```
# Get data directory
pkg <- system.file("extdata", package = "fetch")

# Create catalog
ct <- catalog(pkg, engines$csv)

# View catalog
print(ct)
# data catalog: 6 items
# - Source: C:/packages/fetch/inst/extdata
# - Engine: csv
# - Items:
  # data item 'ADAE': 56 cols 150 rows
  # data item 'ADEX': 17 cols 348 rows
  # data item 'ADPR': 37 cols 552 rows
  # data item 'ADPSGA': 42 cols 695 rows
  # data item 'ADSL': 56 cols 87 rows
  # data item 'ADVS': 37 cols 3617 rows
```

---

print.dinfo *Print a data catalog item*

Description

A class-specific instance of the `print` function for data catalog items. The function prints the info in a summary manner. Use `verbose = TRUE` to print the data info as a list.

Usage

```
## S3 method for class 'dinfo'
print(x, ..., verbose = TRUE)
```

Arguments

`x`	The library to print.
`...`	Any follow-on parameters.
`verbose`	Whether or not to print the info in verbose style. By default, the parameter is FALSE, meaning to print in summary style. Verbose style includes a full data dictionary and printing of all attributes.

Value

The data catalog object, invisibly.

print.specs *Print import specifications*

Description

A function to print the import specification collection.

Usage

```
## S3 method for class 'specs'
print(x, ..., verbose = FALSE)
```

Arguments

x	The specifications to print.
...	Any follow-on parameters to the print function.
verbose	Whether or not to print the specifications in verbose style. By default, the parameter is FALSE, meaning to print in summary style.

Value

The specification object, invisibly.

See Also

Other specs: `import_spec()`, `read.specs()`, `specs()`, `write.specs()`

read.specs *Read import specs from the file system*

Description

A function to read import specifications from the file system. The function accepts a full or relative path to the spec file, and returns the specs as an object. If the `file_path` parameter is passed as a directory name, the function will search for a file with a '.specs' extension and read it.

Usage

```
read.specs(file_path = getwd())
```

Arguments

file_path	The full or relative path to the file system. Default is the current working directory. If the `file_path` is a file name that does not contain the '.specs' file extension, the function will add the extension. If the `file_path` contains a directory name, the function will search the directory for a file with an extension of '.specs'. If more than one file with an extension of '.specs' is founds, the function will generate an error.

Value

The specifications object.

See Also

Other specs: `import_spec()`, `print.specs()`, `specs()`, `write.specs()`

`specs` *Create an Import Spec Collection*

Description

A function to create a collection of import specifications for a data source. These specs can be used on the `catalog` function to correctly assign the data types uniquely for different imported data files. The spec collection is a set of `import_spec` objects identified by name/value pairs. The name corresponds to the name of the input dataset, without file extension. The value is the `import_spec` object to use for that dataset. In this way, you may define different specs for each dataset in your catalog.

The import engines will guess at the data types for any columns that are not explicitly defined in the import specifications. The import spec syntax is the same for all data engines.

Note that the `na` and `trim_ws` parameters on the `specs` function will be applied globally to all files in the library. These global settings can be overridden on the `import_spec` for any particular data file.

Also note that the `specs` collection is defined as an object so it can be stored and reused. See the `write.specs` and `read.specs` functions for additional information on saving and restoring specs.

Usage

```
specs(..., na = c("", "NA"), trim_ws = TRUE)
```

Arguments

`...`	Named input specs. The name should correspond to the file name, without the file extension. The spec is defined as an `import_spec` object. See the `import_spec` function for additional information on parameters for that object.
`na`	A vector of values to be treated as NA. For example, the vector `c('', ' ')` will cause empty strings and single blanks to be converted to NA values. For most file types, empty strings and the string 'NA' (`'', 'NA'`) are considered NA. For SAS® datasets and transport files, a single blank and a single dot `c(" ", ".")` are considered NA. The value of the `na` parameter on the `specs` function can be overridden by the `na` parameter on the `import_spec` function.
`trim_ws`	Whether or not to trim white space from the input data values. Valid values are TRUE, and FALSE. Default is TRUE. The value of the `trim_ws` parameter on the `specs` function can be overridden by the `trim_ws` parameter on the `import_spec` function.

Value

The import spec collection. The class of the object is "specs".

See Also

`catalog` to create a data catalog, `fetch` for retrieving data, and `import_spec` for additional information on defining an import spec.

Other specs: `import_spec()`, `print.specs()`, `read.specs()`, `write.specs()`

Examples

```
# Get sample data directory
pkg <- system.file("extdata", package = "fetch")

# Create import spec
spc <- specs(ADAE = import_spec(TRTSDT = "date=%d%b%Y",
                                TRTEDT = "date=%d%b%Y"),
             ADVS = import_spec(TRTSDT = "character",
                                TRTEDT = "character"))

# Create catalog with specs collection
ct <- catalog(pkg, engines$csv, import_specs = spc)

# Get dictionary for ADAE with Import Spec
d1 <- ct$ADAE

# Observe data types for TRTSDT and TRTEDT are Dates
d1[d1$Column %in% c("TRTSDT", "TRTEDT"), ]
# data item 'ADAE': 56 cols 150 rows
#- Engine: csv
#- Size: 155 Kb
#- Last Modified: 2020-09-18 14:30:22
#   Name Column Class Label Format NAs MaxChar
#13 ADAE TRTSDT  Date  <NA>     NA   1      10
#14 ADAE TRTEDT  Date  <NA>     NA   4      10

# Get dictionary for ADVS with Import Spec
d2 <- ct$ADVS

# Observe data types for TRTSDT and TRTEDT are character
d2[d2$Column %in% c("TRTSDT", "TRTEDT"), ]
# data item 'ADVS': 37 cols 3617 rows
#- Engine: csv
#- Size: 1.1 Mb
#- Last Modified: 2020-09-18 14:30:22
#   Name Column     Class Label Format NAs MaxChar
#16 ADVS TRTSDT character  <NA>     NA  54       9
#17 ADVS TRTEDT character  <NA>     NA 119       9
```

`write.specs` *Write import specs to the file system*

Description

A function to write import specifications to the file system. The function accepts a specifications object and a full or relative path. The function returns the full file path. This function is useful so that you can define import specifications once, and reuse them in multiple programs or across multiple teams.

Usage

```
write.specs(x, dir_path = getwd(), file_name = NULL)
```

Arguments

`x`	A specifications object of class 'specs'.
`dir_path`	A full or relative path to save the specs. Default is the current working directory.
`file_name`	The file name to save to specs, without a file extension. The '.specs' file extension will be added automatically. If no file name is supplied, the function will use the variable name as the file name.

Value

The full file path.

See Also

Other specs: `import_spec()`, `print.specs()`, `read.specs()`, `specs()`

Index