Package 'connector'

May 8, 2026

Title Streamlining Data Access in Clinical Research

Version 1.0.0

Description Provides a consistent interface for connecting R to various data sources including file systems and databases. Designed for clinical research, 'connector' streamlines access to 'ADAM', 'SDTM' for example. It helps to deal with multiple data formats through a standardized API and centralized configuration.

License Apache License (>= 2)

URL https://novonordisk-opensource.github.io/connector/, https://github.com/NovoNordisk-OpenSource/connector/

BugReports https://github.com/NovoNordisk-OpenSource/connector/issues

Depends R (>= 4.1)

Imports arrow, checkmate, cli, DBI, dplyr, fs, glue, haven, jsonlite, lifecycle, purrr, R6 (>= 2.4.0), readr, readxl, rlang, utils, vroom, writexl, yaml, zephyr (>= 0.1.1)

Suggests dbplyr, ggplot2, knitr, rmarkdown, RPostgres, RSQLite, spelling, testthat (>= 3.0.0), tibble, usethis, whirl (>= 0.2.0), withr

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.3.2

NeedsCompilation no

Author Cervan Girard [aut, cre], Aksel Thomsen [aut], Vladimir Obucina [aut], Novo Nordisk A/S [cph]

Maintainer Cervan Girard <cgid@novonordisk.com>

Repository CRAN

Date/Publication 2025-08-19 12:50:07 UTC

Contents

add_datasource *Add a new datasource to a YAML configuration file*

Description

This function adds a new datasource to a YAML configuration file by appending the provided datasource information to the existing datasources.

Usage

```
add_datasource(config_path, name, backend)
```

Arguments

config_path	The file path to the YAML configuration file
name	The name of the new datasource
backend	A named list representing the backend configuration for the new datasource

Value

(invisible) config_path where the configuration have been updated

Examples

```
config <- tempfile(fileext = ".yml")

file.copy(
  from = system.file("config", "_connector.yml", package = "connector"),
  to = config
)

config <- config |>
  add_datasource(
    name = "new_datasource",
    backend = list(type = "connector_fs", path = "new_path")
  )
config
```

add_logs *Add Logging Capability to Connections*

Description

This function adds logging capability to a list of connections by modifying their class attributes. It ensures that the input is of the correct type and registers the necessary S3 methods for logging.

Usage

```
add_logs(connections)
```

Arguments

connections	An object of class connectors(). This should be a list of connection objects to which logging capability will be added.

Details

The function performs the following steps:

1. Checks if the input connections is of class "connectors".
2. Iterates through each connection in the list and prepends the "ConnectorLogger" class.

Value

The modified connections object with logging capability added. Each connection in the list will have the "ConnectorLogger" class prepended to its existing classes.

Examples

```
cnts <- connectors(
  sdtm = connector_fs(path = tempdir())
 )

logged_connections <- add_logs(cnts)

logged_connections
```

---

add_metadata *Add metadata to a YAML configuration file*

Description

This function adds metadata to a YAML configuration file by modifying the provided key-value pair in the metadata section of the file.

Usage

```
add_metadata(config_path, key, value)
```

Arguments

config_path	The file path to the YAML configuration file
key	The key for the new metadata entry
value	The value for the new metadata entry

Value

(invisible) `config_path` where the configuration have been updated

Examples

```
config <- tempfile(fileext = ".yml")

file.copy(
  from = system.file("config", "_connector.yml", package = "connector"),
  to = config
)

config <- config |>
  add_metadata(
    key = "new_metadata",
    value = "new_value"
  )
config
```

`connect` *Connect to datasources specified in a config file*

Description

Based on a configuration file or list this functions creates a `connectors()` object with a Connector for each of the specified datasources.

The configuration file can be in any format that can be read through `read_file()`, and contains a list. If a yaml file is provided, expressions are evaluated when parsing it using `yaml::read_yaml()` with `eval.expr = TRUE`.

See also `vignette("connector")` on how to use configuration files in your project, details below for the required structure of the configuration.

Usage

```
connect(
  config = "_connector.yml",
  metadata = NULL,
  datasource = NULL,
  set_env = TRUE,
  logging = zephyr::get_option("logging", "connector")
)
```

Arguments

config	character path to a connector config file or a list of specifications
metadata	list Replace, add or create elements to the metadata field found in config
datasource	character Name(s) of the datasource(s) to connect to. If NULL (the default) all datasources are connected.
set_env	logical Should environment variables from the yaml file be set? Default is TRUE.
logging	Add logging capability to connectors using add_logs(). When TRUE, all connector operations will be logged to the console and to whirl log HTML files. See log-functions for available logging functions.. Default: FALSE.

Details

The input list can be specified in two ways:

1. A named list containing the specifications of a single connectors object.
2. An unnamed list, where each element is of the same structure as in 1., which returns a nested connectors object. See example below.

Each specification of a single connectors have to have the following structure:

- Only name, metadata, env and datasources are allowed.
- All elements must be named.
- **name** is only required when using nested connectors.
- **datasources** is mandatory.
- **metadata** and **env** must each be a list of named character vectors of length 1 if specified.
- **datasources** must each be a list of unnamed lists.
- Each datasource must have the named character element **name** and the named list element **backend**
- For each connection **backend**.**type** must be provided

Value

connectors

Examples

```
withr::local_dir(withr::local_tempdir("test", .local_envir = .GlobalEnv))
# Create dir for the example in tmpdir
dir.create("example/demo_trial/adam", recursive = TRUE)

# Create a config file in the example folder
config <- system.file("config", "_connector.yml", package = "connector")

# Show the raw configuration file
readLines(config) |>
  cat(sep = "\n")
```

```
# Connect to the datasources specified in it
cnts <- connect(config)
cnts

# Content of each connector

cnts$adam
cnts$sdtm

# Overwrite metadata informations

connect(config, metadata = list(extra_class = "my_class"))

# Connect only to the adam datasource

connect(config, datasource = "adam")

# Connect to several projects in a nested structure

config_nested <- system.file("config", "_nested_connector.yml", package = "connector")

readLines(config_nested) |>
  cat(sep = "\n")

cnts_nested <- connect(config_nested)

cnts_nested

cnts_nested$study1

withr::deferred_run()
```

Connector *General connector object*

Description

This R6 class is a general class for all connectors. It is used to define the methods that all connectors should have. New connectors should inherit from this class, and the methods described below should be implemented.

Methods

Public methods:

- Connector$new()
- Connector$print()
- Connector$list_content_cnt()
- Connector$read_cnt()
- Connector$write_cnt()

- `Connector$remove_cnt()`
- `Connector$clone()`

Method `new()`**:** Initialize the connector with the option of adding an extra class.

Usage:

```
Connector$new(extra_class = NULL)
```

Arguments:

`extra_class` character Extra class to assign to the new connector.

Method `print()`**:** Print method for a connector showing the registered methods and specifications from the active bindings.

Usage:

```
Connector$print()
```

Returns: invisible self.

Method `list_content_cnt()`**:** List available content from the connector. See also list_content_cnt.

Usage:

```
Connector$list_content_cnt(...)
```

Arguments:

`...` Additional arguments passed to the method for the individual connector.

Returns: A character vector of content names

Method `read_cnt()`**:** Read content from the connector. See also read_cnt.

Usage:

```
Connector$read_cnt(name, ...)
```

Arguments:

`name` character Name of the content to read, write, or remove. Typically the table name.

`...` Additional arguments passed to the method for the individual connector.

Returns: R object with the content. For rectangular data a data.frame.

Method `write_cnt()`**:** Write content to the connector.See also write_cnt.

Usage:

```
Connector$write_cnt(x, name, ...)
```

Arguments:

`x` The object to write to the connection

`name` character Name of the content to read, write, or remove. Typically the table name.

`...` Additional arguments passed to the method for the individual connector.

Returns: invisible self.

Method `remove_cnt()`**:** Remove or delete content from the connector. See also remove_cnt.

Usage:

```
Connector$remove_cnt(name, ...)
```

Arguments:

`name` character Name of the content to read, write, or remove. Typically the table name.

`...` Additional arguments passed to the method for the individual connector.

Returns: invisible self.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Connector$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`vignette("customize")` on how to create custom connectors and methods, and concrete examples in ConnectorFS and ConnectorDBI.

Examples

```
# Create connector
cnt <- Connector$new()

cnt

# Standard error message if no method is implemented
cnt |>
  read_cnt("fake_data") |>
  try()

# Connection with extra class
cnt_my_class <- Connector$new(extra_class = "my_class")

cnt_my_class

# Custom method for the extra class
read_cnt.my_class <- function(connector_object) "Hello!"
registerS3method("read_cnt", "my_class", "read_cnt.my_class")

cnt_my_class

read_cnt(cnt_my_class)
```

`connector-options` *Options for connector*

Description

verbosity_level:
Verbosity level for functions in connector. See zephyr::verbosity_level for details.

- Default: `"verbose"`
- Option: `connector.verbosity_level`
- Environment: `R_CONNECTOR_VERBOSITY_LEVEL`

overwrite:
Overwrite existing content if it exists in the connector? See connector-options for details. Default can be set globally with `options(connector.overwrite = TRUE/FALSE)` or environment variable `R_CONNECTOR_OVERWRITE`.

- Default: `FALSE`
- Option: `connector.overwrite`
- Environment: `R_CONNECTOR_OVERWRITE`

logging:
Add logging capability to connectors using `add_logs()`. When `TRUE`, all connector operations will be logged to the console and to whirl log HTML files. See log-functions for available logging functions.

- Default: `FALSE`
- Option: `connector.logging`
- Environment: `R_CONNECTOR_LOGGING`

default_ext:
Default extension to use when reading and writing files when not specified in the file name. E.g. with the default 'csv', files are assumed to be in CSV format if not specified.

- Default: `"csv"`
- Option: `connector.default_ext`
- Environment: `R_CONNECTOR_DEFAULT_EXT`

`ConnectorDBI` *Connector for DBI databases*

Description

Connector object for DBI connections. This object is used to interact with DBI compliant database backends. See the DBI package for which backends are supported.

Details

We recommend using the wrapper function `connector_dbi()` to simplify the process of creating an object of ConnectorDBI class. It provides a more intuitive and user-friendly approach to initialize the ConnectorFS class and its associated functionalities.

Upon garbage collection, the connection will try to disconnect from the database. But it is good practice to call disconnect_cnt when you are done with the connection.

Super class

connector::Connector -> ConnectorDBI

Active bindings

conn The DBI connection. Inherits from DBI::DBIConnector

Methods

Public methods:

- ConnectorDBI$new()
- ConnectorDBI$disconnect_cnt()
- ConnectorDBI$tbl_cnt()
- ConnectorDBI$clone()

Method new(): Initialize the connection

Usage:

```
ConnectorDBI$new(drv, ..., extra_class = NULL)
```

Arguments:

drv Driver object inheriting from DBI::DBIDriver.

... Additional arguments passed to DBI::dbConnect().

extra_class character Extra class to assign to the new connector.

Method disconnect_cnt(): Disconnect from the database. See also disconnect_cnt.

Usage:

```
ConnectorDBI$disconnect_cnt()
```

Returns: invisible self.

Method tbl_cnt(): Use dplyr verbs to interact with the remote database table. See also tbl_cnt.

Usage:

```
ConnectorDBI$tbl_cnt(name, ...)
```

Arguments:

name character Name of the content to read, write, or remove. Typically the table name.

... Additional arguments passed to the method for the individual connector.

Returns: A dplyr::tbl object.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
ConnectorDBI$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
# Create DBI connector
cnt <- ConnectorDBI$new(RSQLite::SQLite(), ":memory:")
cnt

# You can do the same thing using wrapper function connector_dbi()
cnt <- connector_dbi(RSQLite::SQLite(), ":memory:")
cnt
# Write to the database
cnt$write_cnt(iris, "iris")

# Read from the database
cnt$read_cnt("iris") |>
  head()

# List available tables
cnt$list_content_cnt()

# Use the connector to run a query
cnt$conn

cnt$conn |>
  DBI::dbGetQuery("SELECT * FROM iris limit 5")

# Use dplyr verbs and collect data
cnt$tbl_cnt("iris") |>
  dplyr::filter(Sepal.Length > 7) |>
  dplyr::collect()

# Disconnect from the database
cnt$disconnect_cnt()
```

ConnectorFS *Connector for file storage*

Description

The ConnectorFS class is a file storage connector for accessing and manipulating files any file storage solution. The default implementation includes methods for files stored on local or network drives.

Details

We recommend using the wrapper function `connector_fs()` to simplify the process of creating an object of ConnectorFS class. It provides a more intuitive and user-friendly approach to initialize the ConnectorFS class and its associated functionalities.

Super class

connector::Connector -> ConnectorFS

Active bindings

path character Path to the file storage

Methods

Public methods:

- ConnectorFS$new()
- ConnectorFS$download_cnt()
- ConnectorFS$upload_cnt()
- ConnectorFS$create_directory_cnt()
- ConnectorFS$remove_directory_cnt()
- ConnectorFS$upload_directory_cnt()
- ConnectorFS$download_directory_cnt()
- ConnectorFS$tbl_cnt()
- ConnectorFS$clone()

Method new(): Initializes the connector for file storage.

Usage:

```
ConnectorFS$new(path, extra_class = NULL)
```

Arguments:

path character Path to the file storage.

extra_class character Extra class to assign to the new connector.

Method download_cnt(): Download content from the file storage. See also download_cnt.

Usage:

```
ConnectorFS$download_cnt(src, dest = basename(src), ...)
```

Arguments:

src character The name of the file to download from the connector

dest character Path to the file to download to

... Additional arguments passed to the method for the individual connector.

Returns: invisible connector_object.

Method upload_cnt(): Upload a file to the file storage. See also upload_cnt.

Usage:

```
ConnectorFS$upload_cnt(src, dest = basename(src), ...)
```

Arguments:

src character Path to the file to upload

dest character The name of the file to create

... Additional arguments passed to the method for the individual connector.

Returns: invisible self.

Method `create_directory_cnt()`**:** Create a directory in the file storage. See also create_directory_cnt.

Usage:

```
ConnectorFS$create_directory_cnt(name, ...)
```

Arguments:

`name` character The name of the directory to create

`...` Additional arguments passed to the method for the individual connector.

Returns: ConnectorFS object of a newly created directory

Method `remove_directory_cnt()`**:** Remove a directory from the file storage. See also remove_directory_cnt.

Usage:

```
ConnectorFS$remove_directory_cnt(name, ...)
```

Arguments:

`name` character The name of the directory to remove

`...` Additional arguments passed to the method for the individual connector.

Returns: invisible self.

Method `upload_directory_cnt()`**:** Upload a directory to the file storage. See also upload_directory_cnt.

Usage:

```
ConnectorFS$upload_directory_cnt(src, dest = basename(src), ...)
```

Arguments:

`src` character The path to the directory to upload

`dest` character The name of the directory to create

`...` Additional arguments passed to the method for the individual connector.

Returns: invisible self.

Method `download_directory_cnt()`**:** Download a directory from the file storage. See also download_directory_cnt.

Usage:

```
ConnectorFS$download_directory_cnt(src, dest = basename(src), ...)
```

Arguments:

`src` character The name of the directory to download from the connector

`dest` character The path to the directory to download to

`...` Additional arguments passed to the method for the individual connector.

Returns: invisible connector_object.

Method `tbl_cnt()`**:** Use dplyr verbs to interact with the tibble. See also tbl_cnt.

Usage:

```
ConnectorFS$tbl_cnt(name, ...)
```

Arguments:

`name` character Name of the content to read, write, or remove. Typically the table name.

`...` Additional arguments passed to the method for the individual connector.

Returns: A table object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ConnectorFS$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Create file storage connector

folder <- withr::local_tempdir("test", .local_envir = .GlobalEnv)



cnt <- ConnectorFS$new(folder)
cnt

# You can do the same thing using wrapper function connector_fs()
cnt <- connector_fs(folder)
cnt

# List content
cnt$list_content_cnt()

# Write to the connector
cnt$write_cnt(iris, "iris.rds")

# Check it is there
cnt$list_content_cnt()

# Read the result back
cnt$read_cnt("iris.rds") |>
  head()
```

`ConnectorLogger` *Create a New Connector Logger*

Description

Creates a new empty connector logger object of class "ConnectorLogger". This is an internal utility class that initializes a logging structure for connector operations. Logs are added to connectors using `add_logs()`.

Usage

```
ConnectorLogger

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

Arguments

x	object to print
...	parameters passed to the print method

Format

An object of class `ConnectorLogger` of length 0.

Details

Create a New Connector Logger

Value

An S3 object of class "ConnectorLogger" containing:

- An empty list
- Class attribute set to "ConnectorLogger"

Examples

```
logger <- ConnectorLogger
class(logger) # Returns "ConnectorLogger"
str(logger) # Shows empty list with class attribute
```

---

connectors *Collection of connector objects*

Description

Holds a special list of individual connector objects for consistent use of connections in your project.

Usage

```
connectors(...)
```

Arguments

...	Named individual Connector objects

Examples

```
# Create connectors objects

cnts <- connectors(
  sdtm = connector_fs(path = tempdir()),
  adam = connector_dbi(drv = RSQLite::SQLite())
)

# Print for overview

cnts

# Print the individual connector for more information

cnts$sdtm

cnts$adam
```

`connector_dbi` *Create* `dbi` *connector*

Description

Initializes the connector for DBI type of storage. See ConnectorDBI for details.

Usage

```
connector_dbi(drv, ..., extra_class = NULL)
```

Arguments

`drv`	Driver object inheriting from DBI::DBIDriver.
`...`	Additional arguments passed to `DBI::dbConnect()`.
`extra_class`	character Extra class to assign to the new connector.

Details

The `extra_class` parameter allows you to create a subclass of the `ConnectorDBI` object. This can be useful if you want to create a custom connection object for easier dispatch of new s3 methods, while still inheriting the methods from the `ConnectorDBI` object.

Value

A new ConnectorDBI object

Examples

```
# Create DBI connector
cnt <- connector_dbi(RSQLite::SQLite(), ":memory:")
cnt

# Create subclass connection
cnt_subclass <- connector_dbi(RSQLite::SQLite(), ":memory:",
  extra_class = "subclass"
)
cnt_subclass
class(cnt_subclass)
```

---

connector_fs *Create* fs *connector*

Description

Initializes the connector for file system type of storage. See ConnectorFS for details.

Usage

```
connector_fs(path, extra_class = NULL)
```

Arguments

path	character Path to the file storage.
extra_class	character Extra class to assign to the new connector.

Details

The extra_class parameter allows you to create a subclass of the ConnectorFS object. This can be useful if you want to create a custom connection object for easier dispatch of new s3 methods, while still inheriting the methods from the ConnectorFS object.

Value

A new ConnectorFS object

Examples

```
# Create FS connector
cnt <- connector_fs(tempdir())
cnt

# Create subclass connection
cnt_subclass <- connector_fs(
  path = tempdir(),
  extra_class = "subclass"
```

```
)
cnt_subclass
class(cnt_subclass)
```

`create_directory_cnt` *Create a directory*

Description

Generic implementing of how to create a directory for a connector. Mostly relevant for file storage connectors.

- ConnectorFS: Uses `fs::dir_create()` to create a directory at the path of the connector.

Usage

```
create_directory_cnt(connector_object, name, open = TRUE, ...)

## S3 method for class 'ConnectorFS'
create_directory_cnt(connector_object, name, open = TRUE, ...)
```

Arguments

`connector_object`	Connector The connector object to use.
`name`	character The name of the directory to create
`open`	logical Open the directory as a new connector object.
`...`	Additional arguments passed to the method for the individual connector.

Value

invisible connector_object.

Examples

```
# Create a directory in a file storage

folder <- withr::local_tempdir("test", .local_envir = .GlobalEnv)

cnt <- connector_fs(folder)

cnt |>
  list_content_cnt(pattern = "new_folder")

cnt |>
  create_directory_cnt("new_folder")
```

```
# This will return new connector object of a newly created folder
new_connector <- cnt |>
  list_content_cnt(pattern = "new_folder")

cnt |>
  remove_directory_cnt("new_folder")
```

`datasources` *Previously used to extract data sources from connectors*

Description

[Deprecated]. Look for `[list_datasources()]` instead.

Usage

```
datasources(connectors)
```

Arguments

`connectors`	An object containing connectors with a "datasources" attribute.

`disconnect_cnt` *Disconnect (close) the connection of the connector*

Description

Generic implementing of how to disconnect from the relevant connections. Mostly relevant for DBI connectors.

- ConnectorDBI: Uses `DBI::dbDisconnect()` to create a table reference to close a DBI connection.

Usage

```
disconnect_cnt(connector_object, ...)

## S3 method for class 'ConnectorDBI'
disconnect_cnt(connector_object, ...)
```

Arguments

`connector_object`	Connector The connector object to use.
`...`	Additional arguments passed to the method for the individual connector.

Value

invisible connector_object.

Examples

```
# Open and close a DBI connector
cnt <- connector_dbi(RSQLite::SQLite())

cnt$conn

cnt |>
  disconnect_cnt()

cnt$conn
```

`download_cnt` *Download content from the connector*

Description

Generic implementing of how to download files from a connector:

- ConnectorFS: Uses `fs::file_copy()` to copy a file from the file storage to the desired `file`.
- ConnectorLogger: Logs the download operation and calls the underlying connector method.

Usage

```
download_cnt(connector_object, src, dest = basename(src), ...)

## S3 method for class 'ConnectorFS'
download_cnt(connector_object, src, dest = basename(src), ...)

## S3 method for class 'ConnectorLogger'
download_cnt(connector_object, src, dest = basename(src), ...)
```

Arguments

`connector_object`	Connector The connector object to use.
`src`	character Name of the content to read, write, or remove. Typically the table name.
`dest`	character Path to the file to download to or upload from
`...`	Additional arguments passed to the method for the individual connector.

Value

invisible connector_object.

Examples

```
# Download file from a file storage

folder <- withr::local_tempdir("test", .local_envir = .GlobalEnv)

cnt <- connector_fs(folder)

cnt |>
  write_cnt("this is an example", "example.txt")

list.files(pattern = "example.txt")

cnt |>
  download_cnt("example.txt")

list.files(pattern = "example.txt")
readLines("example.txt")

cnt |>
  remove_cnt("example.txt")

# Add logging to a file system connector for downloads
folder <- withr::local_tempdir("test", .local_envir = .GlobalEnv)

cnt <- connectors(data = connector_fs(folder)) |> add_logs()

cnt$data |>
  write_cnt(iris, "iris.csv")

cnt$data |>
  download_cnt("iris.csv", tempfile(fileext = ".csv"))
```

---

download_directory_cnt

Download a directory

Description

Generic implementing of how to download a directory for a connector. Mostly relevant for file storage connectors.

- ConnectorFS: Uses `fs::dir_copy()`.

Usage

```
download_directory_cnt(connector_object, src, dest = basename(src), ...)

## S3 method for class 'ConnectorFS'
download_directory_cnt(connector_object, src, dest = basename(src), ...)
```

Arguments

connector_object	Connector The connector object to use.
src	character The name of the directory to download from the connector
dest	character Path to the directory to download to
...	Additional arguments passed to the method for the individual connector.

Value

invisible connector_object.

Examples

```
# Download a directory to a file storage
folder <- withr::local_tempdir("test", .local_envir = .GlobalEnv)

cnt <- connector_fs(folder)
# Create a source directory
dir.create(file.path(folder, "src_dir"))
writeLines(
  "This is a test file.",
  file.path(folder, "src_dir", "test.txt")
)
# Download the directory
cnt |>
 download_directory_cnt(
   src = "src_dir",
   dest = file.path(folder, "downloaded_dir")
  )
```

---

extract_metadata *Extract metadata from connectors*

Description

This function extracts the "metadata" attribute from a connectors object, with optional filtering to return only a specific metadata field.

Usage

```
extract_metadata(connectors, name = NULL)
```

Arguments

connectors	An object containing connectors with a "metadata" attribute.
name	A character string specifying which metadata attribute to extract. If NULL (default), returns all metadata.

Value

A list containing the metadata extracted from the "metadata" attribute, or the specific attribute value if name is specified.

Examples

```
# A config list with metadata
config <- list(
  metadata = list(
    study = "Example Study",
    version = "1.0"
  ),
  datasources = list(
    list(
      name = "adam",
      backend = list(type = "connector_fs", path = tempdir())
    )
  )
)

cnts <- connect(config)

# Extract all metadata
result <- extract_metadata(cnts)
print(result)

# Extract specific metadata field
study_name <- extract_metadata(cnts, name = "study")
print(study_name)
```

---

list_content_cnt *List available content from the connector*

Description

Generic implementing of how to list all content available for different connectors:

- ConnectorDBI: Uses `DBI::dbListTables()` to list the tables in a DBI connection.
- ConnectorFS: Uses `list.files()` to list all files at the path of the connector.
- ConnectorLogger: Logs the list operation and calls the underlying connector method.

Usage

```
list_content_cnt(connector_object, ...)

## S3 method for class 'ConnectorDBI'
list_content_cnt(connector_object, ...)

## S3 method for class 'ConnectorFS'
list_content_cnt(connector_object, ...)

## S3 method for class 'ConnectorLogger'
list_content_cnt(connector_object, ...)
```

Arguments

connector_object	Connector The connector object to use.
...	Additional arguments passed to the method for the individual connector.

Value

A character vector of content names

Examples

```
# List tables in a DBI database
cnt <- connector_dbi(RSQLite::SQLite())

cnt |>
  list_content_cnt()

# List content in a file storage
folder <- withr::local_tempdir("test", .local_envir = .GlobalEnv)

cnt <- connector_fs(folder)

cnt |>
  list_content_cnt()

#' # Write a file to the file storage
cnt |>
  write_cnt(iris, "iris.csv")

# Only list CSV files using the pattern argument of list.files

cnt |>
  list_content_cnt(pattern = "\\.csv$")

# Add logging to a connector and list contents
folder <- withr::local_tempdir("test", .local_envir = .GlobalEnv)

cnt <- connectors(data = connector_fs(folder)) |> add_logs()
```

```
cnt$data |>
  write_cnt(iris, "iris.csv")

cnt$data |>
  list_content_cnt()
```

`list_datasources` *Extract data sources from connectors*

Description

This function extracts the "datasources" attribute from a connectors object.

Usage

```
list_datasources(connectors)
```

Arguments

`connectors`	An object containing connectors with a "datasources" attribute.

Details

The function uses the `attr()` function to access the "datasources" attribute of the `connectors` object. It directly returns this attribute without any modification.

Value

An object containing the data sources extracted from the "datasources" attribute.

Examples

```
# Connectors object with data sources
cnts <- connectors(
  sdtm = connector_fs(path = tempdir()),
  adam = connector_dbi(drv = RSQLite::SQLite())
)

# Using the function (returns datasources attribute)
result <- list_datasources(cnts)
# Check if result contains datasource information
result$datasources
```

log-functions *Connector Logging Functions*

Description

A comprehensive set of generic functions and methods for logging connector operations. These functions provide automatic logging capabilities for read, write, remove, and list operations across different connector types, enabling transparent audit trails and operation tracking.

Usage

```
log_read_connector(connector_object, name, ...)

## Default S3 method:
log_read_connector(connector_object, name, ...)

log_write_connector(connector_object, name, ...)

## Default S3 method:
log_write_connector(connector_object, name, ...)

log_remove_connector(connector_object, name, ...)

## Default S3 method:
log_remove_connector(connector_object, name, ...)

log_list_content_connector(connector_object, ...)

## S3 method for class 'ConnectorDBI'
log_read_connector(connector_object, name, ...)

## S3 method for class 'ConnectorDBI'
log_write_connector(connector_object, name, ...)

## S3 method for class 'ConnectorDBI'
log_remove_connector(connector_object, name, ...)

## S3 method for class 'ConnectorFS'
log_read_connector(connector_object, name, ...)

## S3 method for class 'ConnectorFS'
log_write_connector(connector_object, name, ...)

## S3 method for class 'ConnectorFS'
log_remove_connector(connector_object, name, ...)
```

Arguments

connector_object	The connector object to log operations for. Can be any connector class (ConnectorFS, ConnectorDBI, ConnectorLogger, etc.)
name	Character string specifying the name or identifier of the resource being operated on (e.g., file name, table name)
...	Additional parameters passed to specific method implementations. May include connector-specific options or metadata.

Details

Connector Logging Functions

The logging system is built around S3 generic functions that dispatch to specific implementations based on the connector class. Each operation is logged with contextual information including connector details, operation type, and resource names.

Value

These are primarily side-effect functions that perform logging. The actual return value depends on the specific method implementation, typically:

- `log_read_connector`: Result of the read operation
- `log_write_connector`: Invisible result of write operation
- `log_remove_connector`: Invisible result of remove operation
- `log_list_content_connector`: List of connector contents

Available Operations

`log_read_connector(connector_object, name, ...)` Logs read operations when data is retrieved from a connector. Automatically called by `read_cnt()` and `tbl_cnt()` methods.

`log_write_connector(connector_object, name, ...)` Logs write operations when data is stored to a connector. Automatically called by `write_cnt()` and `upload_cnt()` methods.

`log_remove_connector(connector_object, name, ...)` Logs removal operations when resources are deleted from a connector. Automatically called by `remove_cnt()` methods.

`log_list_content_connector(connector_object, ...)` Logs listing operations when connector contents are queried. Automatically called by `list_content_cnt()` methods.

Supported Connector Types

Each connector type has specialized logging implementations:

ConnectorFS File system connectors log the full file path and operation type. Example log: `"dataset.csv @ /path/to/data"`

ConnectorDBI Database connectors log driver information and database name. Example log: `"table_name @ driver: SQLiteDriver, dbname: mydb.sqlite"`

Integration with whirl Package

All logging operations use the **whirl** package for consistent log output:

- `whirl::log_read()` - For read operations
- `whirl::log_write()` - For write operations
- `whirl::log_delete()` - For remove operations

See Also

`add_logs` for adding logging capability to connectors, `ConnectorLogger` for the logger class, **whirl** package for underlying logging implementation

Examples

```
# Basic usage with file system connector
logged_fs <- add_logs(connectors(data = connector_fs(path = tempdir())))

# Write operation (automatically logged)
write_cnt(logged_fs$data, mtcars, "cars.csv")
# Output: "cars.csv @ /tmp/RtmpXXX"

#' # Read operation (automatically logged)
data <- read_cnt(logged_fs$data, "cars.csv")
# Output: "dataset.csv @ /tmp/RtmpXXX"

# Database connector example
logged_db <- add_logs(connectors(db = connector_dbi(RSQLite::SQLite(), ":memory:")))

# Operations are logged with database context
write_cnt(logged_db$db, iris, "iris_table")
# Output: "iris_table @ driver: SQLiteDriver, dbname: :memory:"
```

`nested_connectors` *Create a nested connectors object*

Description

This function creates a nested connectors object from the provided arguments.

Usage

```
nested_connectors(...)
```

Arguments

`...`	Any number of connectors object.

Value

A list with class "nested_connectors" containing the provided arguments.

`read_cnt` *Read content from the connector*

Description

Generic implementing of how to read content from the different connector objects:

- ConnectorDBI: Uses `DBI::dbReadTable()` to read the table from the DBI connection.
- ConnectorFS: Uses `read_file()` to read a given file. The underlying function used, and thereby also the arguments available through `...` depends on the file extension.

The file is located using internal function, which searches for files matching the provided name. If multiple files match and no extension is specified, it will use the default extension (configurable via `options(connector.default_ext = "csv")`, defaults to "csv").

- ConnectorLogger: Logs the read operation and calls the underlying connector method.

Usage

```
read_cnt(connector_object, name, ...)

## S3 method for class 'ConnectorDBI'
read_cnt(connector_object, name, ...)

## S3 method for class 'ConnectorFS'
read_cnt(connector_object, name, ...)

## S3 method for class 'ConnectorLogger'
read_cnt(connector_object, name, ...)
```

Arguments

`connector_object`	Connector The connector object to use.
`name`	character Name of the content to read, write, or remove. Typically the table name.
`...`	Additional arguments passed to the method for the individual connector.

Value

R object with the content. For rectangular data a data.frame.

Examples

```
# Read table from DBI database
cnt <- connector_dbi(RSQLite::SQLite())

cnt |>
  write_cnt(iris, "iris")

cnt |>
  list_content_cnt()

cnt |>
  read_cnt("iris") |>
  head()

# Write and read a CSV file using the file storage connector

folder <- withr::local_tempdir("test", .local_envir = .GlobalEnv)

cnt <- connector_fs(folder)

cnt |>
  write_cnt(iris, "iris.csv")

cnt |>
  read_cnt("iris.csv") |>
  head()

# Add logging to a file system connector
folder <- withr::local_tempdir("test", .local_envir = .GlobalEnv)

cnt <- connectors(data = connector_fs(folder)) |> add_logs()

cnt$data |>
  write_cnt(iris, "iris.csv")

cnt$data |>
  read_cnt("iris.csv", show_col_types = FALSE) |>
  head()
```

`read_file` *Read files based on the extension*

Description

`read_file()` is the backbone of all read_cnt methods, where files are read from their source. The function is a wrapper around `read_ext()`, that controls the dispatch based on the file extension.

`read_ext()` controls which packages and functions are used to read the individual file extensions. Below is a list of all the pre-defined methods:

- default: All extensions not listed below is attempted to be read with vroom::vroom()
- txt: readr::read_lines()
- csv: readr::read_csv()
- parquet: arrow::read_parquet()
- rds: readr::read_rds()
- sas7bdat: haven::read_sas()
- xpt: haven::read_xpt()
- yml/yaml: yaml::read_yaml()
- json: jsonlite::read_json()
- excel: readxl::read_excel()

Usage

```
read_file(path, ...)

read_ext(path, ...)

## Default S3 method:
read_ext(path, ...)

## S3 method for class 'txt'
read_ext(path, ...)

## S3 method for class 'csv'
read_ext(path, delim = ",", ...)

## S3 method for class 'parquet'
read_ext(path, ...)

## S3 method for class 'rds'
read_ext(path, ...)

## S3 method for class 'sas7bdat'
read_ext(path, ...)

## S3 method for class 'xpt'
read_ext(path, ...)

## S3 method for class 'yml'
read_ext(path, ...)
```

```
## S3 method for class 'json'
read_ext(path, ...)

## S3 method for class 'xlsx'
read_ext(path, ...)
```

Arguments

path	character() Path to the file.
...	Other parameters passed on the functions behind the methods for each file extension.
delim	Single character used to separate fields within a record.

Value

the result of the reading function

Examples

```
# Read CSV file
temp_csv <- tempfile("iris", fileext = ".csv")
write.csv(iris, temp_csv, row.names = FALSE)
# Read the CSV file using read_ext.csv
read_file(temp_csv, show_col_types = FALSE)
```

---

remove_cnt *Remove content from the connector*

Description

Generic implementing of how to remove content from different connectors:

- ConnectorDBI: Uses DBI::dbRemoveTable() to remove the table from a DBI connection.
- ConnectorFS: Uses fs::file_delete() to delete the file.
- ConnectorLogger: Logs the remove operation and calls the underlying connector method.

Usage

```
remove_cnt(connector_object, name, ...)

## S3 method for class 'ConnectorDBI'
remove_cnt(connector_object, name, ...)

## S3 method for class 'ConnectorFS'
remove_cnt(connector_object, name, ...)
```

```
## S3 method for class 'ConnectorLogger'
remove_cnt(connector_object, name, ...)
```

Arguments

connector_object	Connector The connector object to use.
name	character Name of the content to read, write, or remove. Typically the table name.
...	Additional arguments passed to the method for the individual connector.

Value

invisible connector_object.

Examples

```
# Remove table in a DBI database
cnt <- connector_dbi(RSQLite::SQLite())

cnt |>
  write_cnt(iris, "iris") |>
  list_content_cnt()

cnt |>
  remove_cnt("iris") |>
  list_content_cnt()

# Remove a file from the file storage

folder <- withr::local_tempdir("test", .local_envir = .GlobalEnv)

cnt <- connector_fs(folder)

cnt |>
  write_cnt("this is an example", "example.txt")
cnt |>
  list_content_cnt(pattern = "example.txt")

cnt |>
  read_cnt("example.txt")

cnt |>
  remove_cnt("example.txt")

cnt |>
  list_content_cnt(pattern = "example.txt")

# Add logging to a connector and remove content
folder <- withr::local_tempdir("test", .local_envir = .GlobalEnv)
```

```
cnt <- connectors(data = connector_fs(folder)) |> add_logs()

cnt$data |>
  write_cnt(iris, "iris.csv")

cnt$data |>
  remove_cnt("iris.csv")
```

`remove_datasource` *Remove a datasource from a YAML configuration file*

Description

This function removes a datasource from a YAML configuration file based on the provided name, ensuring that it doesn't interfere with other existing datasources.

Usage

```
remove_datasource(config_path, name)
```

Arguments

`config_path`	The file path to the YAML configuration file
`name`	The name of the datasource to be removed

Value

(invisible) `config_path` where the configuration have been updated

Examples

```
config <- tempfile(fileext = ".yml")

file.copy(
  from = system.file("config", "_connector.yml", package = "connector"),
  to = config
)
# Add a datasource first
config <- config |>
  add_datasource(
    name = "new_datasource",
    backend = list(type = "connector_fs", path = "new_path")
  )
config
# Now remove it
config <- config |>
  remove_datasource("new_datasource")
config
```

`remove_directory_cnt` *Remove a directory*

Description

Generic implementing of how to remove a directory for a connector. Mostly relevant for file storage connectors.

- ConnectorFS: Uses `fs::dir_delete()` to remove a directory at the path of the connector.

Usage

```
remove_directory_cnt(connector_object, name, ...)

## S3 method for class 'ConnectorFS'
remove_directory_cnt(connector_object, name, ...)
```

Arguments

`connector_object`	Connector The connector object to use.
`name`	character The name of the directory to remove
`...`	Additional arguments passed to the method for the individual connector.

Value

invisible connector_object.

Examples

```
# Remove a directory from a file storage

folder <- withr::local_tempdir("test", .local_envir = .GlobalEnv)

cnt <- connector_fs(folder)

cnt |>
  create_directory_cnt("new_folder")

cnt |>
  list_content_cnt(pattern = "new_folder")

cnt |>
  remove_directory_cnt("new_folder") |>
  list_content_cnt(pattern = "new_folder")
```

`remove_metadata` *Remove metadata from a YAML configuration file*

Description

This function removes metadata from a YAML configuration file by deleting the specified key from the metadata section of the file.

Usage

```
remove_metadata(config_path, key)
```

Arguments

`config_path`	The file path to the YAML configuration file
`key`	The key for the metadata entry to be removed

Value

(invisible) `config_path` where the configuration have been updated

Examples

```
config <- tempfile(fileext = ".yml")

file.copy(
  from = system.file("config", "_connector.yml", package = "connector"),
  to = config
)
# Add metadata first
config <- config |>
  add_metadata(
    key = "new_metadata",
    value = "new_value"
  )
config
#' # Now remove it
config <- config |>
  remove_metadata("new_metadata")
config
```

`resource-validation` *Resource Validation System for Connector Objects*

Description

This module provides a flexible validation system to verify that resources required by connector objects exist and are accessible. The validation is performed through S3 method dispatch, allowing each connector class to define its own validation logic while providing a consistent interface.

Usage

```
validate_resource(x)

check_resource(self)

## S3 method for class 'Connector'
check_resource(self)

## S3 method for class 'ConnectorFS'
check_resource(self)
```

Arguments

`x`	Connector object to validate.
`self`	Connector object for method dispatch.

Architecture

The system is built around two main components:

- `validate_resource()`: A dispatcher function that finds and executes the appropriate S3 method based on the connector's class
- `check_resource()`: A generic S3 method that defines the validation interface for all connector types

Method Resolution

The validation process follows this hierarchy:

1. Attempt to find a class-specific method (e.g., `check_resource.ConnectorFS`)
2. If no specific method exists, fall back to the default `check_resource.Connector`
3. Execute the resolved method with appropriate error handling

Available S3 Methods

`check_resource.Connector` Default method that performs no validation. Serves as a safe fallback for connector classes without specific validation needs.

`check_resource.ConnectorFS` Validates file system resources by checking directory existence using `fs::dir_exists()`. Throws informative errors for missing directories.

Implementation Guidelines

When implementing new connector classes with resource validation:

- Define a method following the pattern `check_resource.<YourClass>`
- Return `NULL` on successful validation
- Use `cli::cli_abort()` for validation failures to provide consistent error formatting
- Include `call = rlang::caller_env()` in error calls for proper error context

Error Handling

The validation system provides robust error handling:

- Method resolution failures are handled gracefully with fallback to default
- Validation errors include contextual information about the failing resource
- Error messages use `cli` formatting for consistency across the package

Examples

```
# Basic validation for a file system connector

fs_connector <- try(ConnectorFS$new(path = "doesn_t_exists"), silent = TRUE)
fs_connector
```

---

tbl_cnt *Use dplyr verbs to interact with the remote database table*

Description

Generic implementing of how to create a `dplyr::tbl()` connection in order to use dplyr verbs to interact with the remote database table. Mostly relevant for DBI connectors.

- ConnectorDBI: Uses `dplyr::tbl()` to create a table reference to a table in a DBI connection.
- ConnectorFS: Uses `read_cnt()` to allow redundancy between fs and dbi.

Usage

```
tbl_cnt(connector_object, name, ...)

## S3 method for class 'ConnectorDBI'
tbl_cnt(connector_object, name, ...)

## S3 method for class 'ConnectorFS'
tbl_cnt(connector_object, name, ...)

## S3 method for class 'ConnectorLogger'
tbl_cnt(connector_object, name, ...)
```

Arguments

connector_object	Connector The connector object to use.
name	character Name of the content to read, write, or remove. Typically the table name.
...	Additional arguments passed to the method for the individual connector.

Value

A dplyr::tbl object.

Examples

```
# Use dplyr verbs on a table in a DBI database
cnt <- connector_dbi(RSQLite::SQLite())

iris_cnt <- cnt |>
  write_cnt(iris, "iris") |>
  tbl_cnt("iris")

iris_cnt

iris_cnt |>
  dplyr::collect()

iris_cnt |>
  dplyr::group_by(Species) |>
  dplyr::summarise(
    n = dplyr::n(),
    mean.Sepal.Length = mean(Sepal.Length, na.rm = TRUE)
  ) |>
  dplyr::collect()

# Use dplyr verbs on a table

folder <- withr::local_tempdir("test", .local_envir = .GlobalEnv)

cnt <- connector_fs(folder)

cnt |>
  write_cnt(iris, "iris.csv")

iris_cnt <- cnt |>
  tbl_cnt("iris.csv", show_col_types = FALSE)

iris_cnt

iris_cnt |>
  dplyr::group_by(Species) |>
  dplyr::summarise(
    n = dplyr::n(),
```

```
  mean.Sepal.Length = mean(Sepal.Length, na.rm = TRUE)
)
```

`upload_cnt` *Upload content to the connector*

Description

Generic implementing of how to upload files to a connector:

- ConnectorFS: Uses `fs::file_copy()` to copy the `file` to the file storage.
- ConnectorLogger: Logs the upload operation and calls the underlying connector method.

Usage

```
upload_cnt(
  connector_object,
  src,
  dest = basename(src),
  overwrite = zephyr::get_option("overwrite", "connector"),
  ...
)

## S3 method for class 'ConnectorFS'
upload_cnt(
  connector_object,
  src,
  dest = basename(src),
  overwrite = zephyr::get_option("overwrite", "connector"),
  ...
)

## S3 method for class 'ConnectorLogger'
upload_cnt(
  connector_object,
  src,
  dest = basename(src),
  overwrite = zephyr::get_option("overwrite", "connector"),
  ...
)
```

Arguments

`connector_object`
: Connector The connector object to use.

`src`
: character Path to the file to download to or upload from

dest	character Name of the content to read, write, or remove. Typically the table name.
overwrite	Overwrite existing content if it exists in the connector? See connector-options for details. Default can be set globally with `options(connector.overwrite = TRUE/FALSE)` or environment variable `R_CONNECTOR_OVERWRITE`.. Default: `FALSE`.
...	Additional arguments passed to the method for the individual connector.

Value

invisible connector_object.

Examples

```
# Upload file to a file storage

writeLines("this is an example", "example.txt")

folder <- withr::local_tempdir("test", .local_envir = .GlobalEnv)

cnt <- connector_fs(folder)

cnt |>
  list_content_cnt(pattern = "example.txt")

cnt |>
  upload_cnt("example.txt")

cnt |>
  list_content_cnt(pattern = "example.txt")

cnt |>
  remove_cnt("example.txt")

file.remove("example.txt")

# Add logging to a file system connector for uploads
folder <- withr::local_tempdir("test", .local_envir = .GlobalEnv)

cnt <- connectors(data = connector_fs(folder)) |> add_logs()

# Create a temporary file
temp_file <- tempfile(fileext = ".csv")
write.csv(iris, temp_file, row.names = FALSE)

cnt$data |>
  upload_cnt(temp_file, "uploaded_iris.csv")
```

`upload_directory_cnt` *Upload a directory*

Description

Generic implementing of how to upload a directory for a connector. Mostly relevant for file storage connectors.

- ConnectorFS: Uses `fs::dir_copy()`.

Usage

```
upload_directory_cnt(
  connector_object,
  src,
  dest,
  overwrite = zephyr::get_option("overwrite", "connector"),
  open = FALSE,
  ...
)

## S3 method for class 'ConnectorFS'
upload_directory_cnt(
  connector_object,
  src,
  dest,
  overwrite = zephyr::get_option("overwrite", "connector"),
  open = FALSE,
  ...
)
```

Arguments

`connector_object`	Connector The connector object to use.
`src`	character Path to the directory to upload
`dest`	character The name of the new directory to place the content in
`overwrite`	Overwrite existing content if it exists in the connector? See connector-options for details. Default can be set globally with `options(connector.overwrite = TRUE/FALSE)` or environment variable `R_CONNECTOR_OVERWRITE`.. Default: `FALSE`.
`open`	logical Open the directory as a new connector object.
`...`	Additional arguments passed to the method for the individual connector.

Value

invisible connector_object.

Examples

```
# Upload a directory to a file storage
folder <- withr::local_tempdir("test", .local_envir = .GlobalEnv)

cnt <- connector_fs(folder)
# Create a source directory
dir.create(file.path(folder, "src_dir"))
writeLines(
  "This is a test file.",
  file.path(folder, "src_dir", "test.txt")
)
# Upload the directory
cnt |>
  upload_directory_cnt(
    src = file.path(folder, "src_dir"),
    dest = "uploaded_dir"
  )
```

`use_connector` *Use connector*

Description

Utility function to setup connections with connector in your project:

Creates configuration file (default `_connector.yml`)

See `vignette("connector")` for how to configure the file.

Usage

```
use_connector()
```

`write_cnt` *Write content to the connector*

Description

Generic implementing of how to write content to the different connector objects:

- ConnectorDBI: Uses `DBI::dbWriteTable()` to write the table to the DBI connection.
- ConnectorFS: Uses `write_file()` to Write a file based on the file extension. The underlying function used, and thereby also the arguments available through `...` depends on the file extension.

If no file extension is provided in the `name`, the default extension will be automatically appended (configurable via `options(connector.default_ext = "csv")`, defaults to "csv").

- ConnectorLogger: Logs the write operation and calls the underlying connector method.

Usage

```
write_cnt(
  connector_object,
  x,
  name,
  overwrite = zephyr::get_option("overwrite", "connector"),
  ...
)

## S3 method for class 'ConnectorDBI'
write_cnt(
  connector_object,
  x,
  name,
  overwrite = zephyr::get_option("overwrite", "connector"),
  ...
)

## S3 method for class 'ConnectorFS'
write_cnt(
  connector_object,
  x,
  name,
  overwrite = zephyr::get_option("overwrite", "connector"),
  ...
)

## S3 method for class 'ConnectorLogger'
write_cnt(connector_object, x, name, ...)
```

Arguments

connector_object	Connector The connector object to use.
x	The object to write to the connection
name	character Name of the content to read, write, or remove. Typically the table name.
overwrite	Overwrite existing content if it exists in the connector? See connector-options for details. Default can be set globally with `options(connector.overwrite = TRUE/FALSE)` or environment variable `R_CONNECTOR_OVERWRITE`.. Default: `FALSE`.
...	Additional arguments passed to the method for the individual connector.

Value

invisible connector_object.

Examples

```
# Write table to DBI database
cnt <- connector_dbi(RSQLite::SQLite())

cnt |>
  list_content_cnt()

cnt |>
  write_cnt(iris, "iris")

cnt |>
  list_content_cnt()

# Write different file types to a file storage

folder <- withr::local_tempdir("test", .local_envir = .GlobalEnv)

cnt <- connector_fs(folder)

cnt |>
  list_content_cnt(pattern = "iris")

# rds file
cnt |>
  write_cnt(iris, "iris.rds")

# CSV file
cnt |>
  write_cnt(iris, "iris.csv")

cnt |>
  list_content_cnt(pattern = "iris")

# Add logging to a database connector
cnt <- connectors(data = connector_dbi(RSQLite::SQLite())) |> add_logs()

cnt$data |>
  write_cnt(mtcars, "cars")
```

`write_datasources` *Write datasources attribute into a config file*

Description

Reproduce your workflow by creating a config file based on a connectors object and the associated datasource attributes.

Usage

```
write_datasources(connectors, file)
```

Arguments

connectors	A connectors object with associated "datasources" attribute.
file	path to the config file

Value

A config file with datasource attributes which can be reused in the connect function

Examples

```
folder <- withr::local_tempdir("test", .local_envir = .GlobalEnv)

cnt <- connectors(fs = connector_fs(folder))

# Extract the datasources to a config file
yml_file <- tempfile(fileext = ".yml")
write_datasources(cnt, yml_file)
# Check the content of the file
cat(readLines(yml_file), sep = "\n")
# Reconnect using the new config file
re_connect <- connect(yml_file)
re_connect
```

---

write_file *Write files based on the extension*

Description

`write_file()` is the backbone of all `write_cnt()` methods, where files are written to a connector. The function is a wrapper around `write_ext()` where the appropriate function to write the file is chosen depending on the file extension.

`write_ext()` has methods defined for the following file extensions:

- txt: `readr::write_lines()`
- csv: `readr::write_csv()`
- parquet: `arrow::write_parquet()`
- rds: `readr::write_rds()`
- xpt: `haven::write_xpt()`
- yml/yaml: `yaml::write_yaml()`
- json: `jsonlite::write_json()`
- excel: `writexl::write_xlsx()`

Usage

```
write_file(x, file, overwrite = FALSE, ...)

write_ext(file, x, ...)

## S3 method for class 'txt'
write_ext(file, x, ...)

## S3 method for class 'csv'
write_ext(file, x, delim = ",", ...)

## S3 method for class 'parquet'
write_ext(file, x, ...)

## S3 method for class 'rds'
write_ext(file, x, ...)

## S3 method for class 'xpt'
write_ext(file, x, ...)

## S3 method for class 'yml'
write_ext(file, x, ...)

## S3 method for class 'json'
write_ext(file, x, ...)

## S3 method for class 'xlsx'
write_ext(file, x, ...)
```

Arguments

x	Object to write
file	character() Path to write the file.
overwrite	logical Overwrite existing content if it exists.
...	Other parameters passed on the functions behind the methods for each file extension.
delim	character() Delimiter to use. Default is ",".

Details

Note that write_file() will not overwrite existing files unless overwrite = TRUE, while all methods for write_ext() will overwrite existing files by default.

Value

write_file(): invisible() file.

write_ext(): The return of the functions behind the individual methods.

Examples

```
# Write CSV file
temp_csv <- tempfile("iris", fileext = ".csv")
write_file(iris, temp_csv)
```

Index