Package 'connector'

May 9, 2025

Title Streamlining Data Access in Clinical Research

Version 0.1.1

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, 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, 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 2025-05-09 07:00:02 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 |>
  add_datasource(
    name = "new_datasource",
    backend = list(type = "connector_fs", path = "new_path")
  )
```

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

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

logged_connections <- add_logs(con)
```

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

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 logs to the console as well as to the whirl log html files. 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

```
config <- system.file("config", "_connector.yml", package = "connector")

config

# 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
```

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?

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

logging:

Add logs to the console as well as to the whirl log html files

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

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(name, file = basename(name), ...)
```

Arguments:

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

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

`...` 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(file, name = basename(file), ...)
```

Arguments:

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

`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 `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(dir, name = basename(dir), ...)
```

Arguments:

`dir` character The path to the directory to upload

`name` 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(name, dir = name, ...)
```

Arguments:

`name` character The name of the directory to download

`dir` character The path to the directory to download

`...` 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()
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 S3 class constructor that initializes a logging structure for connector operations.

Usage

```
ConnectorLogger
```

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

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

# Print for overview

con

# Print the individual connector for more information

con$sdtm

con$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()
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` *Extract data sources from connectors*

Description

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

Usage

```
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

```
# Assume we have a 'mock_connectors' object with a 'datasources' attribute
mock_connectors <- structure(list(), class = "connectors")
attr(mock_connectors, "datasources") <- list(source1 = "data1", source2 = "data2")

# Using the function
result <- datasources(mock_connectors)
print(result)
```

`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`.

Usage

```
download_cnt(connector_object, name, file = basename(name), ...)

## S3 method for class 'ConnectorFS'
download_cnt(connector_object, name, file = basename(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.
`file`	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()
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")
```

`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, name, dir = name, ...)

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

Arguments

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

Value

invisible connector_object.

`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.

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, ...)
```

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
cnt <- connector_fs(tempdir())

cnt |>
  list_content_cnt()

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

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

---

list_content_cnt.ConnectorLogger

List contents Operation for ConnectorLogger class

Description

Implementation of the log_read_connector function for the ConnectorLogger class.

Usage

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

Arguments

connector_object	The ConnectorLogger object.
...	Additional parameters.

Value

The result of the read operation.

---

log_list_content_connector *List contents*

Description

This function is a generic for logging the List contents of a connector object. The actual implementation of the logging is determined by the specific method for the connector object's class.

Usage

```
log_list_content_connector(connector_object, ...)
```

Arguments

connector_object	The connector object to log the List contents of.
...	Additional parameters passed to the specific method implementation

Value

The result of the specific method implementation.

log_read_connector *Log Read Connector*

Description

This function is a generic for logging the reading of a connector object. The actual implementation of the logging is determined by the specific method for the connector object's class.

Usage

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

Arguments

connector_object	The connector object to log the reading of.
name	The name of the connector.
...	Additional parameters passed to the specific method implementation

Value

The result of the specific method implementation.

log_read_connector.ConnectorDBI *Log Read Operation for connector dbi*

Description

Implementation of the log_read_connector function for the ConnectorDBI class

Usage

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

Arguments

connector_object	The ConnectorDBI object.
name	The name of the connector.
...	Additional parameters.

log_read_connector.ConnectorFS

Log Read Operation for FS connector

Description

Implementation of the log_read_connector function for the ConnectorFS class.

Usage

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

Arguments

connector_object	The ConnectorFS object.
name	The name of the connector.
...	Additional parameters.

log_read_connector.default

Default Log Read Operation

Description

Default implementation of the log_read_connector function.

Usage

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

Arguments

connector_object	The connector object.
name	The name of the connector.
...	Additional parameters.

log_remove_connector *Log Remove Connector*

Description

This function is a generic for logging the removal of a connector object. The actual implementation of the logging is determined by the specific method for the connector object's class.

Usage

```
log_remove_connector(connector_object, name, ...)
```

Arguments

connector_object	The connector object to log the removal of.
name	The name of the connector.
...	Additional parameters passed to the specific method implementation

Value

The result of the specific method implementation.

log_remove_connector.ConnectorDBI *Log Remove Operation for connector dbi*

Description

Implementation of the log_remove_connector function for the ConnectorDBI class.

Usage

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

Arguments

connector_object	The ConnectorDBI object.
name	The name of the connector.
...	Additional parameters.

log_remove_connector.ConnectorFS *Log Remove Operation for FS connector*

Description

Implementation of the log_remove_connector function for the ConnectorFS class.

Usage

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

Arguments

connector_object	The ConnectorFS object.
name	The name of the connector.
...	Additional parameters.

log_remove_connector.default *Default Log Remove Operation*

Description

Default implementation of the log_remove_connector function.

Usage

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

Arguments

connector_object	The connector object.
name	The name of the connector.
...	Additional parameters.

log_write_connector *Log Write Connector*

Description

This function is a generic for logging the writing of a connector object. The actual implementation of the logging is determined by the specific method for the connector object's class.

Usage

```
log_write_connector(connector_object, name, ...)
```

Arguments

connector_object	The connector object to log the writing of.
name	The name of the connector.
...	Additional parameters passed to the specific method implementation

Value

The result of the specific method implementation.

log_write_connector.ConnectorDBI *Log Write Operation for connector dbi*

Description

Implementation of the log_write_connector function for the ConnectorDBI class.

Usage

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

Arguments

connector_object	The ConnectorDBI object.
name	The name of the connector.
...	Additional parameters.

log_write_connector.ConnectorFS

Log Write Operation for FS connector

Description

Implementation of the log_write_connector function for the ConnectorFS class.

Usage

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

Arguments

connector_object	The ConnectorFS object.
name	The name of the connector.
...	Additional parameters.

log_write_connector.default

Default Log Write Operation

Description

Default implementation of the log_write_connector function.

Usage

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

Arguments

connector_object	The connector object.
name	The name of the connector.
...	Additional parameters.

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.

print.ConnectorLogger *Print Method for ConnectorLogger objects*

Description

This function prints the connector logger.

Usage

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

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

Arguments

x	The connector logger object
...	Additional arguments

Details

This method is designed to be called automatically when `print()` is used on an object of class "ConnectorLogger". It uses `NextMethod()` to call the next appropriate method in the method dispatch chain, allowing for the default or any other custom print behavior to be executed.

Value

The result of the next method in the dispatch chain.

The result of the print operation

See Also

`print`

`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.

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, ...)
```

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()
cnt <- connector_fs(folder)

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

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

---

read_cnt.ConnectorLogger

Log Read Operation for ConnectorLogger class

Description

Implementation of the log_read_connector function for the ConnectorLogger class.

Usage

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

Arguments

connector_object	The ConnectorLogger object.
name	The name of the connector.
...	Additional parameters.

Value

The result of the read operation.

`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_file(temp_csv)
```

---

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.

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, ...)
```

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()
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")
```

`remove_cnt.ConnectorLogger` *Log Remove Operation for ConnectorLogger class*

Description

Implementation of the log_remove_connector function for the ConnectorLogger class.

Usage

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

Arguments

`connector_object`	The ConnectorLogger object.
`name`	The name of the connector.
`...`	Additional parameters.

Value

The result of the remove operation.

`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
)

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

---

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()
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
)

config |>
  add_metadata(
```

```
  key = "new_metadata",
  value = "new_value"
) |>
remove_metadata("new_metadata")
```

`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, ...)
```

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()
cnt <- connector_fs(folder)

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

iris_cnt <- cnt |>
  tbl_cnt("iris.csv")

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.

Usage

```
upload_cnt(
  connector_object,
  file,
  name = basename(file),
```

```
  overwrite = zephyr::get_option("overwrite", "connector"),
  ...
)

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

Arguments

connector_object	Connector The connector object to use.
file	character Path to the file to download to or upload from
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?. 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()
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")
```

`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,
  dir,
  name,
  overwrite = zephyr::get_option("overwrite", "connector"),
  open = FALSE,
  ...
)

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

Arguments

`connector_object`	Connector The connector object to use.
`dir`	character Path to the directory to upload
`name`	character The name of the new directory to place the content in
`overwrite`	Overwrite existing content if it exists in the connector?. 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.

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.

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"),
  ...
)
```

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?. 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()
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")
```

---

write_cnt.ConnectorLogger

Log Write Operation for ConnectorLogger class

Description

Implementation of the log_write_connector function for the ConnectorLogger class.

Usage

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

Arguments

connector_object	The ConnectorLogger object.
x	The data to write.
name	The name of the connector.
...	Additional parameters.

Value

Invisible result of the write operation.

---

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

```
# Connect to the datasources specified in it
config <- system.file("config", "_connector.yml", package = "connector")
cnts <- connect(config)

# Extract the datasources to a config file
yml_file <- tempfile(fileext = ".yml")
write_datasources(cnts, yml_file)
```

```
# 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()`
- sas7bdat: `haven::write_sas()`
- 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