Package ‘mstrio’

March 26, 2020

Type Package

Title Interface for 'MicroStrategy' REST API

Version 11.2.1

Maintainer Piotr Kowal <pkowal@microstrategy.com>

Description Interface for creating data sets and extracting data through the 'MicroStrategy' REST API. Access the demo API at <https://demo.microstrategy.com/MicroStrategyLibrary/api-docs/index.html>.

License Apache License 2.0 | file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 3.4.0)

Imports httr (>= 1.4.1), crul, openssl (>= 1.4.1), jsonlite (>= 1.6), methods, data.table, R6, miniUI, rstudioapi, shinyjs, shiny

Suggests httptest, knitr, rmarkdown, testthat

VignetteBuilder knitr

RoxygenNote 7.0.2


NeedsCompilation no

Author Piotr Kowal [cre], Scott Rigney [aut], Zofia Rogala [ctb], Piotr Czyz [ctb], Michal Drzazga [ctb], Oskar Duda [ctb],
close

Closes a connection with MicroStrategy REST API.

Usage

```r
close(connection)
```

## S4 method for signature 'connection'
close(connection)

Arguments

- `connection` : MicroStrategy REST API connection object returned by `connect_mstr()`

Examples

```r
# Connect to a MicroStrategy environment
con <- connect_mstr(base_url = "https://demo.microstrategy.com/MicroStrategyLibrary/api",
                       username = "user",
                       password = "password",
                       project_name = "Financial Reporting")

# A good practice is to disconnect once you're done
# However, the server will disconnect the session after some time has passed
close(con)
```
connection-class

Description

Base S4 class object containing connection parameters

Slots

- username: Username
- password: Password
- base_url: URL for the REST API server
- project_name: Name of the project to connect to (e.g. "MicroStrategy Tutorial")
- project_id: Project ID corresponding to the chosen project name. This is determined when connecting to the project by name.
- application_code: Code used to identify the client with MicroStrategy.
- web_version: web version.
- iserver_version: iServer version.
- VRCH: Current minimum version supported.
- login_mode: Authentication option. Standard (1) or LDAP (16).
- web_version: web version.
- iserver_version: iServer version.
- version_ok: Both iServer and web version are supported.
- ssl_verify: Default TRUE. Attempts to verify SSL certificates with each request.
- auth_token: Token provided by the I-Server after a successful log in.
- cookies: Cookies returned by the I-Server after a successful log in.

connect_mstr

Create a MicroStrategy REST API connection

Description

Establishes and creates a connection with the MicroStrategy REST API.

Usage

```
connect_mstr(
    base_url, 
    username, 
    password, 
    project_name = NULL, 
    project_id = NULL, 
    login_mode = 1, 
    ssl_verify = TRUE
)
```
Arguments

base_url URL of the MicroStrategy REST API server
username Username
password Password
project_name Name of the project you intend to connect to. Case-sensitive
project_id ID of the project you intend to connect to
login_mode Specifies the authentication mode to use. Supported authentication modes are Standard (1) (default) or LDAP (16)
ssl_verify If TRUE (default), verifies the server’s SSL certificates with each request

Value

A connection object to use in subsequent requests

Examples

# Connect to a MicroStrategy environment
con <- connect_mstr(base_url = "https://demo.microstrategy.com/MicroStrategyLibrary/api",
                    username = "user",
                    password = "password",
                    project_name = "Financial Reporting")

# A good practice is to disconnect once you're done
# In case you forget, the server will disconnect the session after some time has passed
close(con)

Cube Extract a MicroStrategy cube into a R Data.Frame

Description

Access, filter, publish, and extract data from MicroStrategy in-memory cubes

Attributes:
- cube_id: Identifier of a pre-existing cube containing the required data.
- parallel (bool, optional): If True, use asynchronous requests to download data. If False (default), this feature will be disabled.

Public fields

- connection MicroStrategy connection object
- cube_id Identifier of a cube.
Methods

Public methods:

- Cube$new()
- Cube$to_dataframe()
- Cube$apply_filters()
- Cube$clear_filters()
- Cube$get_attr_elements()
- Cube$clone()

Method new():

Usage:
Cube$new(connection, cube_id, parallel = FALSE)

Method to_dataframe():

Usage:
Cube$to_dataframe(
  limit = NULL,
  multi_df = FALSE,
  callback = function(x, y) { }
)

Method apply_filters():

Usage:
Cube$apply_filters(attributes = NULL, metrics = NULL, attr_elements = NULL)

Method clear_filters():

Usage:
Cube$clear_filters()

Method get_attr_elements():

Usage:
Cube$get_attr_elements(verbos = TRUE)

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

Usage:
Cube$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.
Examples

```r
# Create a connection object.
connection = connect_mstr(base_url, username, password, project_name)

# Create a cube object.
my_cube <- Cube$new(connection=conn, cube_id="...")

# See attributes and metrics in the report.
my_cube$attributes
my_cube$metrics
my_cube$attr_elements

# Specify attributes and metrics (columns) to be fetched.
my_cube$apply_filters(attributes = my_report$attributes[1:2],
metrics = my_report$metrics[1:2])

# See the selection of attributes, metrics and attribute elements.
my_cube$selected_attributes
my_cube$selected_metrics
my_cube$selected_attr_elements

# Clear filtering to load a full dataset.
my_cube$clear_filters()

# Fetch data from the Intelligence Server.
my_cube$to_dataframe()

# See the dataframe.
my_cube$dataframe
```

Dataset

Create, update, delete and certify MicroStrategy datasets

Description

When creating a new dataset, provide a dataset name and an optional description. When updating a pre-existing dataset, provide the dataset identifier. Tables are added to the dataset in an iterative manner using `add_table()`.

Public fields

- `connection`  
  MicroStrategy connection object
- `name`  
  Name of the dataset
- `description`  
  Description of the dataset. Must be less than or equal to 250 characters
- `dataset_id`  
  Identifier of a pre-existing dataset. Used when updating a pre-existing dataset
- `verbose`  
  Print API requests to console. Used for debugging
Active bindings
    connection  MicroStrategy connection object

Methods

Public methods:

• Dataset$new()
• Dataset$add_table()
• Dataset$create()
• Dataset$update()
• Dataset$publish()
• Dataset$publish_status()
• Dataset$delete()
• Dataset$certify()
• Dataset$clone()

Method new():
    Usage:
    Dataset$new(
        connection,
        name = NULL,
        description = NULL,
        dataset_id = NULL,
        verbose = FALSE
    )

Method add_table():
    Usage:
    Dataset$add_table(
        name,
        data_frame,
        update_policy,
        to_metric = NULL,
        to_attribute = NULL
    )

Method create():
    Usage:
    Dataset$create(folder_id = NULL, auto_upload = TRUE)

Method update():
    Usage:
    Dataset$update(chunksize = 1e+05)

Method publish():
Usage:
Dataset$publish()

Method publish_status():
Usage:
Dataset$publish_status()

Method delete():
Usage:
Dataset$delete()

Method certify():
Usage:
Dataset$certify()

Method clone(): The objects of this class are cloneable with this method.
Usage:
Dataset$clone(deep = FALSE)

Arguments:
depth Whether to make a deep clone.

Examples

# Create data frames
df1 <- data.frame("id" = c(1, 2, 3, 4, 5),
               "first_name" = c("Jason", "Molly", "Tina", "Jake", "Amy"),
               "last_name" = c("Miller", "Jacobson", "Turner", "Milner", "Cooze"))
df2 <- data.frame("id" = c(1, 2, 3, 4, 5),
               "age" = c(42, 52, 36, 24, 73),
               "state" = c("VA", "NC", "WY", "CA", "CA"),
               "salary" = c(50000, 100000, 75000, 85000, 250000))

# Create a list of tables containing one or more tables and their names
my_dataset <- Dataset$new(connection=conn, name="HR Analysis")
my_dataset$add_table("Employees", df1, "add")
my_dataset$add_table("Salaries", df2, "add")
my_dataset$create()

# By default Dataset$create() will upload the data to the Intelligence Server and publish the
dataset.
# If you just want to create the dataset but not upload the row-level data, use
Dataset$create(auto_upload=FALSE)

# followed by
Dataset$update()
Dataset$publish()
# When the source data changes and users need the latest data for analysis and reporting in
# MicroStrategy, mstrio allows you to update the previously created dataset.

ds <- Dataset$new(connection=conn, dataset_id="...")
ds$add_table(name = "Stores", data_frame = stores_df, update_policy = 'update')
ds$add_table(name = "Sales", data_frame = stores_df, update_policy = 'upsert')
ds$update()
ds$publish()

# By default, the raw data is transmitted to the server in increments of 25,000 rows. On very
# large datasets (>1 GB), it is beneficial to increase the number of rows transmitted to the
# Intelligence Server with each request. Do this with the chunksize parameter:

ds$update(chunksize = 500000)

# If you want to cerfify an existing dataset, use
ds$certify()

---

**Report**

Extract a MicroStrategy report into a R Data.Frame

**Description**

Access, filter, publish, and extract data from in-memory reports. Create a Report object to load basic
information on a report dataset. Specify subset of report to be fetched through Report.apply_filters()
by `microstrategy.Connection()`: report_id: Identifier of a pre-existing report containing the re-
quired data. parallel (bool, optional): If True, use asynchronous requests to download data. If False
(default), this feature will be disabled.

**Public fields**

- **connection**  
  MicroStrategy connection object
- **report_id**  
  Identifier of a report.

**Methods**

**Public methods:**

- Report$new()
- Report$to_dataframe()
- Report$apply_filters()
- Report$clear_filters()
- Report$get_attr_elements()
- Report$clone()
Method new():
Usage:
Report$new(connection, report_id, parallel = FALSE)

Method to_dataframe():
Usage:
Report$to_dataframe(limit = NULL, callback = function(x, y) { })

Method apply_filters():
Usage:
Report$apply_filters(attributes = NULL, metrics = NULL, attr_elements = NULL)

Method clear_filters():
Usage:
Report$clear_filters()

Method get_attr_elements():
Usage:
Report$get_attr_elements(verbos = TRUE)

Method clone(): The objects of this class are cloneable with this method.
Usage:
Report$clone(deep = FALSE)
Arguments:
dee Whether to make a deep clone.

Examples

# Create a connection object.
connection = connect_mstr(base_url, username, password, project_name)

# Create a report object.
my_report <- Report$new(connection, report_id)

# See attributes and metrics in the report.
my_report$attributes
my_report$metrics
my_report$attr_elements

# Specify attributes and metrics (columns) to be fetched.
my_report$apply_filters(attributes = my_report$attributes[1:2],
metrics = my_report$metrics[1:2])

# See the selection of attributes, metrics and attribute elements.
my_report$selected_attributes
my_report$selected_metrics
my_report$selected_attr_elements

# Clear filtering to load a full dataset.
my_report$clear_filters()

# Fetch data from the Intelligence Server.
my_report$to_dataframe()

# See the dataframe.
my_report$dataframe
Index

.connection (connection-class), 3

close, 2
close, connection-method (close), 2
connect_mstr, 3
connection-class, 3
Cube, 4

Dataset, 6

Report, 9