Package 'mstrio'

December 10, 2019

Type Package

Title Interface for 'MicroStrategy' REST API

Version 11.2.0

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), openssl (>= 1.4.1), jsonlite (>= 1.6), methods, R6, miniUI, rstudioapi, shinyjs, shiny

Suggests httptest, knitr, rmarkdown, testthat

VignetteBuilder knitr

RoxygenNote 6.1.1

Collate 'api-authentication.R' 'api-cubes.R' 'api-datasets.R' 'api-projects.R' 'api-reports.R' 'api-misc.R' 'cube.R' 'datasets.R' 'report.R' 'microstrategy.R' 'utils-model.R' 'utils-formjson.R' 'utils-parsejson.R' 'utils-encoder.R' 'utils-filter.R' 'utils-composefilter.R' 'utils-files.R' 'utils-gui.R' 'utils-fetching.R' 'utils-helpers.R' 'utils-update.R' 'server.R' 'app.R'

NeedsCompilation no

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

Filip Godlewski [ctb],
Ignacy Hologa [ctb],
Adam Piotrowski [ctb]

Repository CRAN

Date/Publication 2019-12-10 14:40:03 UTC

R topics documented:

`close` *Closes a connection with MicroStrategy REST API*

Description

Closes a connection with MicroStrategy REST API.

Usage

```
close(connection)

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

Arguments

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

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
# However, the server will disconnect the session after some time has passed
close(con)
```

---

connection-class *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)
```

`create_dataset` *Create an in-memory MicroStrategy dataset (deprecated)*

Description

Creates an in-memory dataset from an R Data.Frame. This function is deprecated. Check out the add_table() & create() method from the Dataset class, which allows for uploading multi-table datasets. `Dataset`

Usage

```
create_dataset(connection, data_frame, dataset_name, table_name,
  to_metric = NULL, to_attribute = NULL, folder_id = NULL,
  description = NULL)

## S4 method for signature 'connection'
create_dataset(connection, data_frame, dataset_name,
  table_name, to_metric = NULL, to_attribute = NULL,
  folder_id = NULL, description = NULL)
```

Arguments

`connection`	MicroStrategy REST API connection object
`data_frame`	R Data.Frame from which an in-memory dataset will be created
`dataset_name`	Name of the in-memory dataset
`table_name`	Name of the table to create within the dataset
`to_metric`	(optional) A vector of column names from the Data.Frame to format as metrics in the dataset. By default, numeric types are formatted as metrics while character and date types are formatted as attributes. For example, a column of integer-like strings ("1", "2", "3") would appear as an attribute in the newly created dataset. If the intent is to format this data as a metric, provide the corresponding column name as `to_metric=c('myStringIntegers')`
`to_attribute`	(optional) Logical opposite of `to_metric`. Helpful for formatting an integer-based row identifier as a primary key in the dataset
`folder_id`	(optional) ID of the shared folder that the dataset should be created within. If `NULL`, defaults to the user's My Reports folder.
`description`	(optional) Description of the dataset. Must be less than or equal to 250 characters.

Value

Unique identifiers of the dataset and table within the newly created dataset. Required for `update_dataset()`

Examples

```
df <- iris

# Create a primary key
df$ID <- as.character(row.names(df))

# Remove periods and other special characters due to their
# special role in MicroStrategy. But, "_" is ok.
names(df) <- c("Sepal_Length", "Sepal_Width", "Petal_Length", "Petal_Width", "Species", "ID")

# Create the dataset
mydf <- create_dataset(connection = conn,
                       data_frame = df,
                       dataset_name = "IRIS",
                       table_name = "IRIS")

# You can specify special treatment for columns within the data frame.
# This will convert the character-formatted row ID's to a MicroStrategy metric
mydf <- create_dataset(connection = conn,
                       data_frame = df,
                       dataset_name = "IRIS",
                       table_name = "IRIS",
                       to_metric = c("ID"))

# This will convert 'Sepal_Length' and 'Sepal_Width' to attributes
mydf <- create_dataset(connection = conn,
                       data_frame = df,
                       dataset_name = "IRIS",
                       table_name = "IRIS",
                       to_attribute = c("Sepal_Length", "Sepal_Width"))
```

Cube *Extract a MicroStrategy cube into a R Data.Frame*

Description

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

Usage

```
Cube
```

Format

An object of class `R6ClassGenerator` of length 25.

Fields

`connection` MicroStrategy connection object

`cube_id` Identifier of a cube.

Examples

```
# 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, and delete 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()‘.

Usage

```
Dataset
```

Format

An object of class `R6ClassGenerator` of length 25.

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

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)
```

`Filter` *Pull MicroStrategy cubes (full or filtered)*

Description

Pass ids of selected objects (attributes, metrics and elements).

Usage

```
Filter
```

Format

An object of class `R6ClassGenerator` of length 25.

Fields

`attributes` List of ids for selected attributes.

`metrics` List of ids for selected metrics.

`attr_elements` List of ids for selected attribute elements.

`get_cube` *Extract a MicroStrategy cube into a R Data.Frame (deprecated)*

Description

Extracts the contents of a MicroStrategy cube into a R Data.Frame. This function is deprecated. Check out the to_dataframe() from the Cube class. `Cube`

Usage

```
get_cube(connection, cube_id, offset = 0, limit = 1000)

## S4 method for signature 'connection'
get_cube(connection, cube_id, offset = 0,
  limit = 1000)
```

Arguments

connection	MicroStrategy REST API connection object
cube_id	Unique ID of the cube you wish to extract information from
offset	(optional) To extract all data from the report, use 0 (default)
limit	(optional) Used to control data extract behavior on datasets with a large number of rows. The default is 1000. As an example, if the dataset has 50,000 rows, `get_cube()` will incrementally extract all 50,000 rows in 1,000 row chunks. Depending on system resources, a higher limit (e.g. 10,000) may reduce the total time required to extract the entire dataset

Value

R Data.Frame containing the cube contents

Examples

```
# Extract the contents of a cube into an R Data.Frame
my_cube <- get_cube(connection = conn,
                    cube_id = "5E2501A411E8756818A50080EF4524C9")

# Extract the contents in larger 'chunks' using limit.
# May require add'l server processing time.
# As a rule-of-thumb, aim for a limit setting around 10%
# to 20% of the total number of rows in the cube.
my_cube <- get_cube(connection = conn,
                    cube_id = "5E2501A411E8756818A50080EF4524C9",
                    limit = 100000)

# You can also set limit to -1. Use this only on smaller reports.
my_cube <- get_cube(connection = conn,
                    cube_id = "5E2501A411E8756818A50080EF4524C9",
                    limit = -1)
```

---

get_report *Extracts the contents of a report into a R Data.Frame (deprecated)*

Description

Extracts the contents of a MicroStrategy report into a R Data.Frame. This function is deprecated. Check out the to_dataframe() from the Report class. Report

Usage

```
get_report(connection, report_id, offset = 0, limit = 1000)

## S4 method for signature 'connection'
get_report(connection, report_id, offset = 0,
  limit = 1000)
```

Arguments

connection	MicroStrategy REST API connection object
report_id	Unique ID of the report you wish to extract information from
offset	(optional) To extract all data from the report, use 0 (default)
limit	(optional) Used to control data extract behavior on datasets with a large number of rows. The default is 1000. As an example, if the dataset has 50,000 rows, `get_report()` will incrementally extract all 50,000 rows in 1,000 row chunks. Depending on system resources, a higher limit (e.g. 10,000) may reduce the total time required to extract the entire dataset

Value

R Data.Frame containing the report contents

Examples

```
# Extract the contents of a report into an R Data.Frame
my_report <- get_report(connection = conn,
                        report_id = "5E2501A411E8756818A50080EF4524C9")

# Extract the contents in larger 'chunks' using limit.
# May require add'l server processing time.
# As a rule-of-thumb, aim for a limit setting around 10%
# to 20% of the total number of rows in the report.
my_report <- get_report(connection = conn,
                        report_id = "5E2501A411E8756818A50080EF4524C9",
                        limit = 100000)

# You can also set limit to -1. Use this only on smaller reports.
my_report <- get_report(connection = conn,
                        report_id = "5E2501A411E8756818A50080EF4524C9",
                        limit = -1)
```

`Model` *Create the definition of multi-table and single-table datasets*

Description

Create the definition of a dataset containing one or more tables. The definition includes the name and description of the dataset and the name and description of each table, attribute, and metric within the dataset.

Usage

```
Model
```

Format

An object of class `R6ClassGenerator` of length 25.

Fields

`tables` List containing lists of data.frames and corresponding table names

`name` Name of the dataset

`description` Description of the data set. Must be less than or equal to 250 characters

`folder_id` ID of the shared folder that the dataset should be created within. If NULL, defaults to the user's My Reports folder

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
tables = list(list("table_name" = "employee_id",
                   "data_frame" = df1),
              list("table_name" = "employee_data",
                   "data_frame" = df2))

# Generate the data model
model <- Model$new(tables=tables, name="Employees", description="Employee Analytics Data")
model_info <- model$get_model()
```

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() and Report.clear_filters() . Fetch dataset through Report.to_dataframe() method.

Usage

```
Report
```

Format

An object of class `R6ClassGenerator` of length 25.

Fields

`connection` MicroStrategy connection object

`report_id` Identifier of a report.

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
```

`update_dataset` *Update a previously created dataset (deprecated)*

Description

Updates a previously created MicroStrategy dataset with an R Data.Frame. This function is deprecated. Check out the add_table() & update() & publish() method from the Dataset class, which allows for updating multi-table datasets. `Dataset`

Usage

```
update_dataset(connection, data_frame, dataset_id, table_id, table_name,
  update_policy)

## S4 method for signature 'connection'
update_dataset(connection, data_frame, dataset_id,
  table_name, update_policy)
```

Arguments

`connection`	MicroStrategy REST API connection object
`data_frame`	R Data.Frame to use to update an in-memory dataset
`dataset_id`	Identifier of the dataset to update, provided by `create_dataset()`
`table_id`	Not used. Identifier of the table to update within the dataset, provided by `create_dataset()`
`table_name`	Name of the table to update within the dataset
`update_policy`	Update operation to perform. One of 'add' (inserts new, unique rows), 'update' (updates data in existing rows and columns), 'upsert' (updates existing data and inserts new rows), 'replace' (similar to truncate and load, replaces the existing data with new data)

Examples

```
df <- iris

# Create a primary key
df$ID <- as.character(row.names(df))

# Remove periods and other special characters due to their
# special role in MicroStrategy. But, "_" is ok.
```

```
names(df) <- c("Sepal_Length", "Sepal_Width", "Petal_Length", "Petal_Width", "Species", "ID")

# Create the dataset
mydf <- create_dataset(connection = conn,
                       data_frame = df,
                       dataset_name = "IRIS",
                       table_name = "IRIS")

# Add new rows to the dataset with update policy "add"
df2 <- df[sample(nrow(df), 5), ]
df2[, 'ID'] <- as.character(nrow(df) + seq(1:5))
update_dataset(connection = conn, data_frame = df2,
               dataset_id = mydf$datasetID,
               table_id = mydf$tableID,
               table_name = mydf$name,
               update_policy = 'add')

# Update existing data in the dataset with update policy "update"
df$Sepal_Length <- df$Sepal_Length + runif(nrow(df))
df$Petal_Width <- df$Sepal_Length + rnorm(nrow(df))
update_dataset(connection = conn, data_frame = df,
               dataset_id = mydf$datasetID,
               table_id = mydf$tableID,
               table_name = mydf$name,
               update_policy = 'update')

# Update and add new rows to the dataset with update policy "upsert"
df$Sepal_Length <- df$Sepal_Length + runif(nrow(df))
df$Petal_Width <- df$Sepal_Length + rnorm(nrow(df))
df2 <- df[sample(nrow(df), 5), ]
df2[, 'ID'] <- as.character(nrow(df) + seq(1:5))
df <- rbind(df, df2)
update_dataset(connection = conn,
               data_frame = df,
               dataset_id = mydf$datasetID,
               table_id = mydf$tableID,
               table_name = mydf$name,
               update_policy = 'upsert')

# Truncate and load new data into the dataset with update policy "replace"
df[] <- lapply(df, sample)
update_dataset(connection = conn, data_frame = df,
               dataset_id = mydf$datasetID,
               table_id = mydf$tableID,
               table_name = mydf$name,
               update_policy = 'replace')

# It is possible to update a dataset if it wasn't created in this session or by another client.
# Simply provide the dataset ID and table IDs to this function as characters.
df[] <- lapply(df, sample)  # shuffle contents of the dataframe
update_dataset(connection = conn, data_frame = df,
               dataset_id = "5E2501A411E8756818A50080EF4524C9",
               table_id = "F0DA816816432E448F1105327C119596",
```

```
table_name = "IRIS",
update_policy = 'replace')
```

Index