Package ‘mstrio’

December 21, 2020

Type Package

Title Interface for 'MicroStrategy' REST API

Version 11.3.0.1

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, rstudioapi, shinyjs, shiny

Suggests httptest, knitr, rmarkdown, testthat

VignetteBuilder knitr

RoxygenNote 7.1.0

Collate 'api-authentication.R' 'api-cubes.R' 'api-datasets.R'
  'api-projects.R' 'api-reports.R' 'api-misc.R' 'connection.R'
  'cube.R' 'dataset.R' 'report.R' 'utils-model.R'
  'utils-formjson.R' 'utils-parser.R' 'utils-encoder.R'
  'utils-errors.R' 'utils-filter.R' 'utils-files.R' 'utils-gui.R'
  'utils-import.R' 'utils-helpers.R' 'utils-export.R' 'server.R'
  'app.R'

NeedsCompilation no

Author Scott Rigney [aut],
  Piotr Kowal [cre],
  Piotr Czyz [ctb],
  Ignacy Hologa [ctb],
  Michal Drzazga [ctb],
  Oskar Duda [ctb],
Connection

Filip Godlewski [ctb],
Karolina Sachmata [ctb],
Adam Piotrowski [ctb],
Zofia Rogala [ctb],
Wojciech Antonczyk [ctb]

Repository CRAN

Date/Publication 2020-12-21 13:40:03 UTC

R topics documented:

- Connection .......................................................... 2
- Cube ................................................................. 5
- Dataset .............................................................. 8
- Report ............................................................... 12

Index 16

<table>
<thead>
<tr>
<th>Connection</th>
<th>Connect to the MicroStrategy environment.</th>
</tr>
</thead>
</table>

Description

allows to establish, renew, check status and close the connection with MicroStrategy Intelligence Server.

Public fields

- base_url URL of the MicroStrategy REST API server.
- username Your username.
- password Your password.
- project_name Name of the connected MicroStrategy Project. One of project name or project id is necessary.
- project_id ID of the connected MicroStrategy Project. One of project name or project id is necessary.
- login_mode Specifies authentication mode to use. Standard = 1 (default) or LDAP = 16.
- ssl_verify If True (default), verifies the server’s SSL certificates with each request.
- web_version The current web version
- iserver_version The current I-Server version
- auth_token The authentication token returned by the I-Server
- cookies Cookies
- identity_token Identity token for delegated session. Used for connection initialized by GUI.
- verbose If True (default), displays additional messages.
Methods

**Public methods:**
- `Connection$new()`
- `Connection$connect()`
- `Connection$delegate()`
- `Connection$get_identity_token()`
- `Connection$close()`
- `Connection$renew()`
- `Connection$status()`
- `Connection$clone()`

**Method new():** Establishes new connection with MicroStrategy Intelligence Server.

**Usage:**
```perl
Connection$new(
    base_url,
    username = NULL,
    password = NULL,
    project_name = NULL,
    project_id = NULL,
    login_mode = 1,
    ssl_verify = TRUE,
    proxies = NULL,
    identity_token = NULL,
    verbose = TRUE
)
```

**Arguments:**
- `base_url` URL of the MicroStrategy REST API server.
- `username` Your username.
- `password` Your password.
- `project_name` Name of the connected MicroStrategy Project. One of project name or project id is necessary.
- `project_id` ID of the connected MicroStrategy Project. One of project name or project id is necessary.
- `login_mode` Specifies authentication mode to use. Standard = 1 (default) or LDAP = 16.
- `ssl_verify` If True (default), verifies the server's SSL certificates with each request.
- `proxies` If NULL (default) proxy is not defined. To set proxy use formula: (<username>:<password>@)<ip_address>:<port>.
- `identity_token` Identity token for delegated session. Used for connection initialized by GUI.
- `verbose` If True, displays additional messages. FALSE by default.

**Returns:** A new "Connection" object.

**Method connect():** Establishes new connection with MicroStrategy Intelligence Server, or renews active connection.

**Usage:**
Connection$connect()

**Method** delegate(): Delegates identity token to get authentication token and connect to MicroStrategy Intelligence Server

*Usage:*
Connection$delegate()

**Method** get_identity_token(): Gets identity token using existing authentication token

*Usage:*
Connection$get_identity_token()

**Method** close(): Closes a connection with MicroStrategy REST API.

*Usage:*
Connection$close()

**Method** renew(): Renews connection with MicroStrategy REST API.

*Usage:*
Connection$renew()

**Method** status(): Displays status of the connection with MicroStrategy REST API.

*Usage:*
Connection$status()

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

*Usage:*
Connection$clone(deep = FALSE)

*Arguments:*
deep Whether to make a deep clone.

**Examples**

```r
## Not run:
# Create a connection object.
connection = Connection$new(base_url, username, password, project_name)

# Connect or renew connection.
connection$connect()

# Check connection status.
connection$status()

# Renew connection to reset timeout counter.
connection$renew()

# Close connection.
connection$close()

## End(Not run)
```
**Cube**

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

---

**Description**

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

Create a `Cube` object to load basic information on a cube dataset. Specify subset of cube to be fetched through `apply_filters()` and `clear_filters()`. Fetch dataset through `to_dataframe()` method.

**Public fields**

- **connection** MicroStrategy connection object
- **cube_id** Identifier of a report.
- **parallel** If TRUE, downloads cube data asynchronously. FALSE by default.
- **name** Cube name.
- **owner_id** ID of Cube owner.
- **path** Exact path of the cube location.
- **last_modified** Date of latest Cube modification.
- **size** Cube size.
- **status** Cube status.
- **attributes** Cube attributes.
- **metrics** Cube metrics
- **attr_elements** Cube attribute elements.
- **selected_attributes** Attributes selected for filtering.
- **selected_metrics** Metrics selected for filtering.
- **selected_attr_elements** Attribute elements selected for filtering.
- **dataframe** Dataframe containing data fetched from the Cube.
- **dataframe_list** List of dataframes split to match tables in Cube.
- **instance_id** Identifier of an instance if cube instance has been already initialized.

**Methods**

**Public methods:**

- `Cube$new()`
- `Cube$to_dataframe()`
- `Cube$apply_filters()`
- `Cube$clear_filters()`
- `Cube$get_attr_elements()`
- `Cube$update()`
- `Cube$save_as()`
- **Cube$clone()**

**Method new():** Initialize an instance of a cube.

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

**Arguments:**
- `connection` MicroStrategy connection object. See Connection class.
- `cube_id` Identifier of a pre-existing cube containing the required data.
- `instance_id` Identifier of an instance if cube instance has been already initialized, NULL by default.
- `parallel` (bool, optional): If True, utilize optimal number of threads to increase the download speed. If False (default), this feature will be disabled.

**Method to_dataframe():** Extract contents of a cube into a R Data Frame.

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

**Arguments:**
- `limit` (int, optional): Used to control data extraction behaviour on cubes with a large number of rows. By default the limit is calculated automatically. If TRUE, overrides automatic limit.
- `multi_df` If True (default), returns a list of dataframes resembling the table structure of the cube. If FALSE, returns one dataframe.
- `callback` used by the GUI to extract the progress information.

**Returns:** Dataframe with data fetched from the given Cube.

**Method apply_filters():** Apply filters on the cube data so only the chosen attributes, metrics, and attribute elements are retrieved from the Intelligence Server.

**Usage:**
Cube$apply_filters(
    attributes = NULL,
    metrics = NULL,
    attr_elements = NULL,
    operator = "In" )

**Arguments:**
- `attributes` (list or None, optional): ID numbers of attributes to be included in the filter. If list is empty, no attributes will be selected and metric data will be aggregated.
- `metrics` (list or None, optional): ID numbers of metrics to be included in the filter. If list is empty, no metrics will be selected.
- `attr_elements` (list or None, optional): Attributes’ elements to be included in the filter.
operator (character, optional): Supported view filter operators are either "In" or "NotIn". This defines whether data will include ("In") or exclude ("NotIn") the supplied attr_elements values.

**Method clear_filters():** Clear previously set filters, allowing all attributes, metrics, and attribute elements to be retrieved.

*Usage:*
Cube$clear_filters()

**Method get_attr_elements():** Load all attribute elements of the Cube. Accessible via Cube$attr_elements. Fetching attribute elements will also allow for validating attribute elements by the filter object.

*Usage:*
Cube$get_attr_elements(limit = 50000, verbose = TRUE)

*Arguments:*
verbose If TRUE, displays list of attribute elements.

**Method update():** Update single-table cube easily with the data frame stored in the Cube instance (cube$dataframe). Before the update, make sure that the data frame has been modified.

*Usage:*
Cube$update(update_policy = "update")

*Arguments:*
update_policy (character) 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), or 'replace' (replaces the existing data with new data).

**Method save_as():** Creates a new single-table cube with the data frame stored in the Cube instance (cube$dataframe). Before the update, make sure that the data exists.

*Usage:*
Cube$save_as(name, description = NULL, folder_id = NULL, table_name = NULL)

*Arguments:*
name (character): Name of the dataset. Must be less than or equal to 250 characters.
description (character, optional): Description of the dataset. Must be less than or equal to 250 characters.
folder_id ID of the shared folder that the dataset should be created within. If 'None', defaults to the user's My Reports folder.
table_name (character, optional) Name of the table. If NULL, the first table name of the original cube will be used.

**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.
## Not run:

```r
# Create a connection object.
connection = Connection$new(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
```

## End(Not run)

### 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
- **folder_id** If specified the dataset will be saved in this folder
dataset_id  Identifier of a pre-existing dataset. Used when updating a pre-existing dataset
owner_id  Owner ID
path  Cube path
modification_time  Last modification time, "yyyy-MM-dd HH:mm:ss" in UTC
size  Cube size
cube_state  Cube status, for example, 0=unpublished, 1=publishing, 64=ready
verbose  If True (default), displays additional messages.

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(): Interface for creating, updating, and deleting MicroStrategy in-memory datasets.

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

Arguments:
connection  MicroStrategy connection object returned by ‘Connection$New()’.
name  (character): Name of the dataset.
description  (character, optional): Description of the dataset. Must be less than or equal to 250 characters.
dataset_id  (character, optional): Identifier of a pre-existing dataset. Used when updating a pre-existing dataset.
verbose  Setting to control the amount of feedback from the I-Server.

Details:  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()’.

Returns:  A new ‘Datasets’ object
**Method** `add_table()`: Add a data.frame to a collection of tables which are later used to update the MicroStrategy dataset

**Usage:**
```
Dataset$add_table(
    name,  name (character): Logical name of the table that is visible to users of the dataset in MicroStrategy.
    data_frame,  data_frame (‘data.frame’): R data.frame to add or update.
    update_policy,  update_policy (character): 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), or 'replace' (replaces the existing data with new data).
    to_metric = NULL,  to_metric (optional, vector): By default, R numeric data types are treated as metrics in the MicroStrategy dataset while character and date types are treated as attributes. For example, a column of integer-like strings ("1", "2", "3") would, by default, be an attribute in the newly created dataset. If the intent is to format this data as a metric, provide the respective column name as a character vector in 'to_metric' parameter.
    to_attribute = NULL
)
```

**Arguments:**
- `name` (character): Logical name of the table that is visible to users of the dataset in MicroStrategy.
- `data_frame` (‘data.frame’): R data.frame to add or update.
- `update_policy` (character): 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), or 'replace' (replaces the existing data with new data).
- `to_metric` (optional, vector): By default, R numeric data types are treated as metrics in the MicroStrategy dataset while character and date types are treated as attributes. For example, a column of integer-like strings ("1", "2", "3") would, by default, be an attribute in the newly created dataset. If the intent is to format this data as a metric, provide the respective column name as a character vector in 'to_metric' parameter.
- `to_attribute` (optional, vector): Logical opposite of 'to_metric'. Helpful for formatting an integer-based row identifier as a primary key in the dataset.

**Details:** Add tables to the dataset in an iterative manner using ‘add_table()’.

**Method** `create()`: Create a new dataset.

**Usage:**
```
Dataset$create(
    folder_id = NULL,  folder_id ID of the shared folder that the dataset should be created within. If ‘None’, defaults to the user’s My Reports folder.
    auto_upload = TRUE,  auto_upload (default TRUE) If True, automatically uploads the data to the I-Server. If False, simply creates the dataset definition but does not upload data to it.
    auto_publish = TRUE,  auto_publish (default TRUE) If True, automatically publishes the data used to create the dataset definition. If False, simply creates the dataset but does not publish it. To publish the dataset, data has to be uploaded first.
    chunksize = 1e+05
)
```

**Arguments:**
- `folder_id` ID of the shared folder that the dataset should be created within. If ‘None’, defaults to the user’s My Reports folder.
- `auto_upload` (default TRUE) If True, automatically uploads the data to the I-Server. If False, simply creates the dataset definition but does not upload data to it.
- `auto_publish` (default TRUE) If True, automatically publishes the data used to create the dataset definition. If False, simply creates the dataset but does not publish it. To publish the dataset, data has to be uploaded first.
- `chunksize` (int, optional) Number of rows to transmit to the I-Server with each request when uploading.

**Method** `update()`: Updates an existing dataset with new data.
Usage:
Dataset$update(chunksize = 1e+05, auto_publish = TRUE)

Arguments:
chunksize (int, optional): Number of rows to transmit to the I-Server with each request when uploading.

auto_publish (default TRUE) If True, automatically publishes the data. If False, data will be uploaded but the cube will not be published

Method publish(): Publish the uploaded data to the selected dataset. A dataset can be published just once.
Usage:
Dataset$publish()

Method publish_status(): Check the status of data that was uploaded to a dataset.
Usage:
Dataset$publish_status()

Returns: Response status code

Method delete(): Delete a dataset that was previously created using the REST API.
Usage:
Dataset$delete()

Returns: Response object from the Intelligence Server acknowledging the deletion process.

Method certify(): Certify a dataset that was previously created using the REST API
Usage:
Dataset$certify()

Returns: Response object from the Intelligence Server acknowledging the certification process.

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

Arguments:
deep Whether to make a deep clone.

Examples
## Not run:
# 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(auto_publish=TRUE)

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

# By default, the raw data is transmitted to the server in increments of 100,000 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 certify an existing dataset, use
ds$certify()

## End(Not run)

---

### 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.
Public fields

connection  MicroStrategy connection object
report_id  Identifier of a report.
parallel  If TRUE, downloads report data asynchronously. FALSE by default.
name  Report name.
attributes  Report attributes.
metrics  Report metrics
attr_elements  Report attribute elements.
selected_attributes  Attributes selected for filtering.
selected_metrics  Metrics selected for filtering.
selected_attr_elements  Attribute elements selected for filtering.
dataframe  Dataframe containing data fetched from the Report.
cross_tab  boolean for filtering cross tabbed reports logic
cross_tab_filters  view filters for cross tab reports
instance_id  Identifier of an instance if report instance has been already initialized.

Methods

Public methods:

• Report$new()
• Report$to_dataframe()
• Report$apply_filters()
• Report$clear_filters()
• Report$get_attr_elements()
• Report$clone()

Method  new():  Initialize an instance of a report.

Usage:
Report$new(connection, report_id, instance_id = NULL, parallel = FALSE)

Arguments:
connection  MicroStrategy connection object. See Connection class.
report_id  Identifier of a pre-existing report containing the required data.
instance_id  Identifier of an instance if report instance has been already initialized, NULL by default.
parallel  (bool, optional):  If True, utilize optimal number of threads to increase the download speed. If False (default), this feature will be disabled.

Method  to_dataframe():  Extract contents of a Report into a R Data Frame.

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

- **limit** (int, optional): Used to control data extraction behaviour on report with a large number of rows. By default, the limit is calculated automatically. If TRUE, overrides automatic limit.

- **callback** used by the GUI to extract the progress information

Returns: Dataframe with data fetched from the given Report.

**Method apply_filters()**: Apply filters on the report data so only the chosen attributes, metrics, and attribute elements are retrieved from the Intelligence Server.

**Usage**:

```r
Report$apply_filters(
  attributes = NULL,
  metrics = NULL,
  attr_elements = NULL,
  operator = "In"
)
```

Arguments:

- **attributes** (list or None, optional): ID numbers of attributes to be included in the filter. If list is empty, no attributes will be selected and metric data will be aggregated.

- **metrics** (list or None, optional): ID numbers of metrics to be included in the filter. If list is empty, no metrics will be selected.

- **attr_elements** (list or None, optional): Attributes’ elements to be included in the filter.

- **operator** (character, optional): Supported view filter operators are either "In" or "NotIn". This defines whether data will include ("In") or exclude ("NotIn") the supplied attr_elements values.

**Method clear_filters()**: Clear previously set filters, allowing all attributes, metrics, and attribute elements to be retrieved.

**Usage**:

```r
Report$clear_filters()
```

**Method get_attr_elements()**: Load all attribute elements of the Report. Accessible via Report$attr_elements. Fetching attribute elements will also allow for validating attribute elements by the filter object.

**Usage**:

```r
Report$get_attr_elements(limit = 50000, verbose = TRUE)
```

Arguments:

- **limit** How many rows of data to fetch per request.
- **verbose** If TRUE, displays list of attribute elements.

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

**Usage**:

```r
Report$clone(deep = FALSE)
```

Arguments:

- **deep** Whether to make a deep clone.
Examples

```r
## Not run:
# Create a connection object.
connection = Connection$new(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

## End(Not run)
```
Index

Connection, 2
Cube, 5
Dataset, 8
Report, 12