Package: sasctl (via r-universe)

May 29, 2026

Title The sasctl package enables easy communication between the SAS Viya platform APIs and the R runtime

Version 0.8.1.9000

Author Eduardo Hellas

Maintainer Eduardo Hellas <eduardo.hellas@sas.com>

Description The sasctl package enables easy communication between the SAS Viya platform APIs and the R runtime.

License file LICENSE

Encoding UTF-8

LazyData false

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Suggests future, furrr, testthat (>= 3.0.0), httptest, knitr, rmarkdown, tidymodels, xgboost, rstudioapi

Config/testthat/edition 3

Imports jsonlite, httr, uuid, ROCR, utils, reshape2, methods, base64enc, glue

URL https://sassoftware.github.io/r-sasctl/

BugReports https://github.com/sassoftware/r-sasctl/issues

VignetteBuilder knitr

Config/pak/sysreqs libicu-dev libssl-dev

Repository https://pinduzera.r-universe.dev

Date/Publication 2026-04-28 19:54:07 UTC

RemoteUrl https://github.com/sassoftware/r-sasctl

RemoteRef HEAD

RemoteSha a5932499f50cf99071740515a5ce276fb2cc206b

Contents

`add_model_content` *Add model content*

Description

Add model content

Usage

```
add_model_content(session, file, model, role = NULL, exact = TRUE, ...)
```

Arguments

`session`	viya_connection object, obtained through `session` function
`file`	path to file
`model`	`MMmodel` object, model ID or model name. If name, will try to find a single model with exact name match. See `exact` parameter
`role`	file role, such as "scoreResource" and "score"
`exact`	the filter query should use "contains" for partial match or "eq" for exact match
`...`	pass to `sasctl::vPOST()` function

Value

a `MMcontent` class list

Examples

```
## Not run:
my_model <- get_model(sess, "MyModel")

myContent <- add_model_content(sess, file = "my_fancy_file.R", model = my_model)
myContent

## End(Not run)
```

add_model_version *Add model version*

Description

Add model version

Usage

```
add_model_version(session, model, exact = TRUE, minor = FALSE, ...)
```

Arguments

session	viya_connection object, obtained through session function
model	MMmodel object, project ID or model name. If name, will try to find a single model with exact name match. See exact parameter
exact	the filter query should use "contains" for partial match or "eq" for exact match
minor	if TRUE, new minor version is created, by default a major version is created.
...	additional parameters to be passed to httr::POST such as httr::add_headers

Value

A data.frame with the list of projects

Examples

```
## Not run:
my_model <- get_model(sess, model = "MyModel")

nvmodel <- add_model_version(sess,  my_model)

nvmodel

## End(Not run)
```

calculateFitStat *Write dmcas_fistat Json*

Description

Calculates fit statistics from user data and writes it to a JSON file for importing into the common model repository.

Usage

```
calculateFitStat(
  targetName,
  targetPredicted,
  validadedf = NULL,
  traindf = NULL,
  testdf = NULL,
  type = "binary",
  targetEventValue = 1,
  path = "./",
  label.ordering = c(0, 1),
  cutoff = 0.5,
  noFile = FALSE
)
```

Arguments

`targetName`	target variable column name (actuals)
`targetPredicted`	target variable column name. When `type = "binary"` it should be a probability.
`validadedf`	`data.frame` where the first column in the yActual (labels/value) and the second is yPrediction (target probability)
`traindf`	`data.frame` where the first column in the yActual (labels/value) and the second is yPrediction (target probability)
`testdf`	`data.frame` where the first column in the yActual (labels/value) and the second is yPrediction (target probability)
`type`	`"binary"` or `"interval"`
`targetEventValue`	if `type = "binary"` target class name for fit stat reference, if model is nominal, all other class will be counted as "not target"
`path`	default to current work dir
`label.ordering`	The default ordering (cf.details) of the classes can be changed by supplying a vector containing the negative and the positive class label. See `ROCR::prediction()`
`cutoff`	cutoff to be used for calculation of miss classification for binary
`noFile`	if you don't want to write to a file, only the output

Value

- `list` that reflects the 'dmcas_fitstat.json'
- 'dmcas_fitstat.json' file written to `path`

Examples

```
df <- data.frame(label = sample(c(1,0), 6000, replace = TRUE),
                 prob = runif(6000),
```

```
                  partition = rep_len(1:3, 6000))

calculateFitStat(targetName = "label",
                  targetPredicted = "prob",
                  df[df$partition == 1, ],
                  df[df$partition == 2, ],
                  df[df$partition == 3, ],
                  noFile = TRUE)


df2 <- data.frame(actual = rnorm(6000, 1000, 100),
                   predicted = rnorm(6000, 1000, 100),
                   partition = rep_len(1:3, 6000))

calculateFitStat(targetName = "actual",
                  targetPredicted = "predicted",
                  df2[df2$partition == 1, ],
                  df2[df2$partition == 2, ],
                  df2[df2$partition == 3, ],
                  type = "interval",
                  noFile = TRUE)
```

`calculateLiftStat` *Write dmcas_lift Json*

Description

Calculates the lift curves from user data and writes to a JSON file for importing into the common model repository. Binary response only.

Usage

```
calculateLiftStat(
  targetName,
  targetPredicted,
  validadedf = NULL,
  traindf = NULL,
  testdf = NULL,
  targetEventValue = 1,
  path = "./",
  noFile = FALSE
)
```

Arguments

`targetName`	target variable column name (actuals)
`targetPredicted`	target variable predicted probability column name

validadedf	data.frame where the first column in the yActual (labels/value) and the second is yPrediction (target probability)
traindf	data.frame where the first column in the yActual (labels/value) and the second is yPrediction (target probability)
testdf	data.frame where the first column in the yActual (labels/value) and the second is yPrediction (target probability)
targetEventValue	target class name for ROC reference, if model is nominal, all other class will be counted as "not target"
path	default to current work dir
noFile	if you don't want to write to a file, only the output

Value

- list that reflects the 'dmcas_roc.json'
- 'dmcas_roc.json' file written to path

Examples

```
df <- data.frame(label = sample(c(1,0), 6000, replace = TRUE),
                 prob = runif(6000),
           partition = rep_len(1:3, 6000)) ## partition will be ignored since it is 3rd column

calculateLiftStat(targetName = "label",
                  targetPredicted = "prob",
                  df[df$partition == 1, ],
                  df[df$partition == 2, ],
                  df[df$partition == 3, ],
                  noFile = TRUE)
```

---

calculateROCStat *Write dmcas_roc Json*

Description

Calculates the ROC curve from user data and writes it to a JSON file for importing into the common model repository. Binary response only.

Usage

```
calculateROCStat(
  targetName,
  targetPredicted,
  validadedf = NULL,
```

```
  traindf = NULL,
  testdf = NULL,
  targetEventValue = 1,
  label.ordering = c(0, 1),
  path = "./",
  noFile = FALSE
)
```

Arguments

`targetName`	target variable column name (actuals)
`targetPredicted`	target variable predicted probability column name
`validadedf`	`data.frame` where the first column in the yActual (labels/value) and the second is yPrediction (target probability)
`traindf`	`data.frame` where the first column in the yActual (labels/value) and the second is yPrediction (target probability)
`testdf`	`data.frame` where the first column in the yActual (labels/value) and the second is yPrediction (target probability)
`targetEventValue`	target class name for ROC reference, if model is nominal, all other class will be counted as "not target"
`label.ordering`	The default ordering (cf.details) of the classes can be changed by supplying a vector containing the negative and the positive class label. See `ROCR::prediction()`
`path`	default to current work dir
`noFile`	if you don't want to write to a file, only the output

Value

- `list` that reflects the 'dmcas_roc.json'
- 'dmcas_roc.json' file written to `path`

Examples

```
df <- data.frame(label = sample(c(1,0), 6000, replace = TRUE),
                 prob = runif(6000),
           partition = rep_len(1:3, 6000)) ## partition will be ignored since it is 3rd column

calculateROCStat(targetName = "label",
                 targetPredicted = "prob",
                 df[df$partition == 1, ],
                 df[df$partition == 2, ],
                 df[df$partition == 3, ],
                 noFile = TRUE)
```

codegen *SAS standard score code generation Generic Function* **(EXPERIMENTAL)**

Description

EXPERIMENTAL STATE - MAY NOT WORK AS INTENDED

Score code will only be generated successfully for supported models. Other models and frameworks will be added in due time. Use `create_scoreSample()` to get a structure sample

Disclaimer: The score code that is generated is designed to be a working template for an R model, but is not guaranteed to work out of the box for scoring, publishing, or validating the model.

Usage

```
codegen(
  model,
  path,
  rds,
  libs,
  inputs,
  output_as_df = TRUE,
  add_target_name = TRUE,
  ...
)

## S3 method for class 'lm'
codegen(
  model,
  path = "scoreCode.R",
  rds = "model.rds",
  libs = c(),
  inputs = NULL,
  output_as_df = TRUE,
  add_target_name = TRUE,
  ...
)

## S3 method for class 'glm'
codegen(
  model,
  path = "scoreCode.R",
  rds = "model.rds",
  libs = c(),
  inputs = NULL,
  output_as_df = TRUE,
  add_target_name = TRUE,
```

```
  cutoff = 0.5,
  ...
)

## S3 method for class 'workflow'
codegen(
  model,
  path = "scoreCode.R",
  rds = "model.rds",
  libs = c(),
  inputs = NULL,
  output_as_df = TRUE,
  add_target_name = TRUE,
  referenceLevel = NULL,
  ...
)
```

Arguments

`model`	model object (lm, glm, tidymodels workflow, ...)
`path`	file name and path to write
`rds`	.rds file name to be called
`libs`	vector of libraries to be added to the code. Some may be guessed from the type.
`inputs`	define inputs as the passed vector instead of guessed
`output_as_df`	logical; when `TRUE` (default) the score function returns a `data.frame`, otherwise a list
`add_target_name`	logical; when `TRUE` (default) includes `<<target>> = <<target>>` in the code output
`...`	to be passes to individual code generators
`cutoff`	classification probability cutoff
`referenceLevel`	reference level for a factor target value

Value

a code string

Methods (by class)

- `codegen(lm)`: Code generator for `lm` class models
- `codegen(glm)`: generator for `glm` class models, specifically logistic regression
- `codegen(workflow)`: generator for tidymodels `workflow` class models

Examples

```
## Not run:
# SAS viya doesn't play nice with variables with '.' in the names
colnames(iris) <- gsub("\\.", "_", colnames(iris))

# simple regression
model <- lm(Petal.Length ~ ., data = iris)

codegen(model)

## End(Not run)
```

`convert_to_pmml42` *Convert pmml 4.x to 4.2*

Description

Converts a pmml header text file from 4.x version to 4.2.

Usage

```
convert_to_pmml42(file_in, file_out)
```

Arguments

`file_in`	path to a .pmml file
`file_out`	path to write the converted .pmml file

Details

NOTE: As of SAS Viya 2025.9, it is no longer required to convert PMML 4.x to 4.2

Value

nothing

Examples

```
## Not run:
hmeq <- read.csv("https://support.sas.com/documentation/onlinedoc/viya/exampledatasets/hmeq.csv",
                 stringsAsFactors = TRUE)

hmeq[hmeq == ""] <- NA
hmeq <- na.omit(hmeq)
hmeq$BAD <- as.factor(hmeq$BAD)

model1 <- glm(BAD ~ ., hmeq, family = binomial("logit"))
summary(model1)
```

```
XML::saveXML(pmml::pmml(model1, model.name = "General_Regression_Model",
                        app.name = "Rattle/PMML",
                        description = "Linear Regression Model"),
             "dev/my_model44.pmml")

convert_to_pmml42("my_model.pmml", "my_model_conv.pmml")

## End(Not run)
```

`create_project` *Create a project*

Description

Returns a sasctl MMproject object from Model Manager

Usage

```
create_project(
  session,
  name,
  description = NULL,
  model_function = NULL,
  input_vars = NULL,
  output_vars = NULL,
  image = NULL,
  additional_parameters = NULL,
  ...
)
```

Arguments

`session`	viya_connection object, obtained through `session` function
`name`	The name of the project
`description`	The description of the project.
`model_function`	The project model function of the project. Valid values: analytical, classification, cluster, forecasting, prediction, Text categorization, Text extraction, Text sentiment, Text topics, transformation
`input_vars`	`data.frame` with the input data sample to configure the project variables.
`output_vars`	`data.frame` with the output data sample to configure the project variables.
`image`	Image URI to be used as project cover
`additional_parameters`	`list` of parameters and lists to be added to the payload
`...`	additional parameters to be passed to `httr::POST` such as `httr::add_headers`

Value

A `data.frame` with the list of projects

Examples

```
## Not run:
new_project <- create_project(sess, name = "ModelProj",
                                    description = "My fancy project",
                                    model_function = "classification")
new_project

## End(Not run)
```

`create_scoreSample` *Create Score Code template*

Description

Creates an R file in the path with an example. The file structure are as follows: For official documentation go to Scoring R models documentation

Usage

```
create_scoreSample(path = ".", openFile = TRUE)
```

Arguments

`path`	path to create file, default current working directory
`openFile`	automatically open file for editing

Details

- The file should start with a function with all the input variables which SAS Viya will use to insert data
- Then it is followed by a comment line `#output: outvar1, outvar2` which is case sensitive and must follow that structure so SAS can receive the function output properly.
- If you are using a previously created model, it should be read, we recommend `.rda` format, but could be a `pmml` file or other format that suits you, just make sure that it is properly classified as scoring resource when using inside SAS Model Manager.
- You then can use any logic to score the model or just an arbitrary R code.
- To pass the information back to SAS it must return a list of the variables defined at the beginning of the script.

Value

nothing

Examples

```
## Not run:
create_scoreSample()

## End(Not run)

# SAS does not expect the following outputs necessarily
# but if you follow that structure it will play nice with other SAS Features

# EM_CLASSIFICATION - Predicted for target
# EM_EVENTPROBABILITY - Probability target=1
# EM_PROBABILITY - Probability of Classification
# I_<<target>> - eg.: I_BAD - Into: BAD
# I_<<target>><<level>> eg.: I_BAD1 - predicted level
```

`delete_client` *Delete a client*

Description

Delete a client

Usage

```
delete_client(session, client)
```

Arguments

`session`	viya_connection object, obtained through `session` function
`client`	`sasClient` object or client ID.

Value

A `httr::response` object.

Examples

```
## Not run:
new_client <- register_client(sess, 'my_client', 'my_s3cr3t!')
delete_client(sess, "my_client")

## End(Not run)
```

delete_masmodule *Delete a module/model and steps*

Description

Delete a module/model published on MAS.

Usage

```
delete_masmodule(session, module, exact = TRUE)
```

Arguments

session	viya_connection object, obtained through session function
module	MASmodule object, module/model name. If name, will try to find a single module with exact name match. See exact parameter
exact	the filter query should use "contains" for partial match or "eq" for exact match

Value

A httr::response object.

Examples

```
## Not run:
deleted_module <- delete_masmodule(sess, module = "ModuleName")
deleted_module

## End(Not run)
```

delete_model *Delete a model*

Description

delete a model from Model Manager

Usage

```
delete_model(session, model, exact = TRUE)
```

Arguments

session	viya_connection object, obtained through session function
model	MMmodel or MMmodelVersion object, model ID or model name. If name, will try to find a single model with exact name match. See exact parameter
exact	the filter query should use "contains" for partial match or "eq" for exact match

Value

A httr::response object.

Examples

```
## Not run:
my_model <- get_model(sess, "MyModel")

delete_model(sess, my_model)

## End(Not run)
```

---

delete_model_contents *Delete a model content*

Description

delete model from Manager

Usage

```
delete_model_contents(session, model, content, exact = TRUE)
```

Arguments

session	viya_connection object, obtained through session function
model	MMmodel object, model ID or model name. If name, will try to find a single model with exact name match. See exact parameter
content	MMmodelContentList obtained from list_model_contents(), if missing will delete all files will be deleted.
exact	the filter query should use "contains" for partial match or "eq" for exact match

Value

A httr::response object.

Examples

```
## Not run:
my_model <- get_model(sess, "MyModel")

delete_model_contents(sess, my_model)

## End(Not run)
```

`delete_project` *Delete a project*

Description

Delete a project and all associated models and resources

Usage

```
delete_project(session, project, exact = TRUE)
```

Arguments

`session`	viya_connection object, obtained through `session` function
`project`	`MMproject` object, project ID or project name. If name, will try to find a single project with exact name match. See `exact` parameter
`exact`	the filter query should use "contains" for partial match or "eq" for exact match

Value

A `httr::response` object.

Examples

```
## Not run:
new_project <- create_project(sess, name = "ModelProj",
                                    description = "My fancy project",
                                    model_function = "classification")

delete_project(sess, new_project)

## End(Not run)
```

`diagnosticsJson` *Write all diagnostic Json files*

Description

Calculates and writes fit statistics, roc and lift for binary and fit statistics for interval.

Usage

```
diagnosticsJson(
  targetName,
  targetPredicted,
  validadedf = NULL,
  traindf = NULL,
  testdf = NULL,
  type = "binary",
  targetEventValue = 1,
  cutoff = 0.5,
  label.ordering = c(0, 1),
  path = "./",
  noFile = FALSE
)
```

Arguments

`targetName`	target variable name (actuals)
`targetPredicted`	target variable probability column name
`validadedf`	`data.frame` where the first column in the yActual (labels/value) and the second is yPrediction (target probability)
`traindf`	`data.frame` where the first column in the yActual (labels/value) and the second is yPrediction (target probability)
`testdf`	`data.frame` where the first column in the yActual (labels/value) and the second is yPrediction (target probability)
`type`	`"binary"` or `"interval"`
`targetEventValue`	if `type = "binary"` target class name for fit stat reference, if model is nominal, all other class will be counted as "not target"
`cutoff`	cutoff to be used for calculation of miss classification for binary
`label.ordering`	The default ordering (cf.details) of the classes can be changed by supplying a vector containing the negative and the positive class label. See `ROCR::prediction()`
`path`	default to current work dir
`noFile`	if you don't want to write to a file, only the output

Value

- `list` of lists that reflects the 'dmcas_fitstat.json', 'dmcas_roc.json' and 'dmcas_lift.json'
- 'dmcas_fitstat.json', 'dmcas_roc.json' and 'dmcas_lift.json' files written to `path`

See Also

All parameters are passed to `calculateLiftStat()`, `calculateLiftStat()` and `calculateLiftStat()` for matching parameters.

Examples

```
df <- data.frame(label = sample(c(1,0), 6000, replace = TRUE),
                 prob = runif(6000),
           partition = rep_len(1:3, 6000)) ## partition will be ignored since it is 3rd column

diagnosticsJson(df[df$partition == 1, ],
                df[df$partition == 2, ],
                df[df$partition == 3, ],
                targetName = "label",
                targetPredicted = "prob",
                noFile = TRUE
                )
```

`format_data_json` *Format Data.Frame rows to json format*

Description

Viya MAS requires a very specific json format which is the goal of this function to create

Usage

```
format_data_json(df, scr = FALSE, scr_batch = FALSE, metadata_columns = NULL)
```

Arguments

`df`	data frame to be transformed in JSON format rows
`scr`	boolean, if `TRUE` will write the new json format with metadata
`scr_batch`	boolean, if `TRUE` will write the batch format JSON for SCR, metadata is ignored
`metadata_columns`	columns names to be used as metadata. If scr is set to `FALSE`, metatada_columns is ignored

Value

a vector of JSON strings or a single json string when `scr_batch` is set to `TRUE`

Examples

```
json_output <- format_data_json(mtcars)
json_output

json_output <- format_data_json(mtcars, scr = TRUE)
json_output

json_output <- format_data_json(mtcars, scr_batch = TRUE)
jsonlite::prettify(json_output)
```

---

get_client *Get a client*

Description

Returns a single sasctl `MMclient` from SAS Model Manager

Usage

```
get_client(session, client, ...)
```

Arguments

`session`	viya_connection object, obtained through `session` function
`client`	`sasClient` object or client ID.
`...`	additional parameters to be passed to `httr::GET` such as `httr::add_headers`

Value

`list` with sasctl attribute

Examples

```
## Not run:
my_client <- get_client(sess, client = ModelProj)
my_client

## End(Not run)
```

get_destination *Get a publishing destination by name*

Description

Returns a publishing destination

Usage

```
get_destination(session, name, ...)
```

Arguments

session	viya_connection object, obtained through session function
name	destination name
...	additional parameters to be passed to httr::GET such as httr::add_headers

Value

A data.frame with the list of projects

Examples

```
## Not run:
destination <- get_destination(sess, 'maslocal')
destination

## End(Not run)
```

get_masmodule *Get a MAS module/model and steps*

Description

Returns a single module MASmodule object with module and steps information

Usage

```
get_masmodule(session, module, id = NULL, exact = TRUE, ...)
```

Arguments

session	viya_connection object, obtained through session function
module	MASmodule object, module name. If name, will try to find a single model with exact name match. See exact parameter
id	module id, will replace module parameter if given
exact	the filter query should use "contains" for partial match or "eq" for exact match
...	additional parameters to be passed to httr::GET such as httr::add_headers

Value

list with sasctl attribute

Examples

```
## Not run:
my_module <- get_masmodule(sess, model = "name")
my_module

## End(Not run)
```

---

get_model *Get a model*

Description

Returns a single sasctl MMmodel from SAS Model Manager

Usage

```
get_model(session, model, exact = TRUE, ...)
```

Arguments

session	viya_connection object, obtained through session function
model	MMmodel or MMmodelVersion object, project ID or model name. If name, will try to find a single model with exact name match. See exact parameter
exact	the filter query should use "contains" for partial match or "eq" for exact match
...	additional parameters to be passed to httr::GET such as httr::add_headers

Value

list with sasctl attribute

Examples

```
## Not run:
my_model <- get_model(sess, model = MMmodel)

my_model

## End(Not run)
```

`get_project` *Get a project*

Description

Returns a single sasctl `MMProject` from SAS Model Manager

Usage

```
get_project(session, project, exact = TRUE, ...)
```

Arguments

`session`	viya_connection object, obtained through `session` function
`project`	`MMproject` object, project ID or project name. If name, will try to find a single project with exact name match. See `exact` parameter
`exact`	the filter query should use "contains" for partial match or "eq" for exact match
`...`	additional parameters to be passed to `httr::GET` such as `httr::add_headers`

Value

`list` with sasctl attribute

Examples

```
## Not run:
my_project <- get_project(sess, project = ModelProj)

my_project

## End(Not run)
```

list_clients *List Clients*

Description

Returns a list of clients

Usage

```
list_clients(
  session,
  start = 1,
  count = 100,
  filter = NULL,
  exact = FALSE,
  ...
)
```

Arguments

session	viya_connection object, obtained through session function
start	the index of the first project to return
count	The number of results per page. The default is 100.
filter	character string of client_id name to be filtered
exact	boolean, If the filter query should use "co" for partial match or "eq" for exact match
...	additional parameters to be passed to httr::GET such as httr::add_headers

Value

A data.frame of sasclientList class with the list of clients

Examples

```
## Not run:
clients <- list_clients(sess, filter = list(createdBy = "creatorUser", name = "projectName"))
clients

## End(Not run)
```

`list_destinations` *List Publish Destinations*

Description

Returns a list of Publish Destinations

Usage

```
list_destinations(
  session,
  start = 0,
  limit = 10,
  filters = list(),
  exact = FALSE,
  ...
)
```

Arguments

`session`	viya_connection object, obtained through `session` function
`start`	the index of the first project to return
`limit`	maximum number of projects to return
`filters`	list of of names vectors for filter parameters (createdBy, modifiedBy, name). By default it will use the `contains` operation. For more info visit `https://developer.sas.com/apis/rest`
`exact`	boolean, If the filter query should use "contains" for partial match or "eq" for exact match
`...`	additional parameters to be passed to `httr::GET` such as `httr::add_headers`

Value

A `data.frame` with the list of projects

Examples

```
## Not run:
destinations <- list_destinations(sess)
destinations

## End(Not run)
```

`list_model_contents` *List Models*

Description

Returns a list of models from Model Manager

Usage

```
list_model_contents(session, model, start = 0, limit = 20, exact = FALSE, ...)
```

Arguments

`session`	viya_connection object, obtained through `session` function
`model`	`MMmodel` or `MMmodelVersion` object, model ID or model name. If name, will try to find a single model with exact name match. See `exact` parameter
`start`	the index of the first content to return
`limit`	maximum number of models to return
`exact`	boolean, If the filter query should use "contains" for partial match or "eq" for exact match
`...`	additional parameters to be passed to `httr::GET` such as `httr::add_headers`

Value

A `MMmodelContentList` list with the list of contents

Examples

```
## Not run:
models <- list_models(sess, filter = list(createdBy = "creatorUser", name = "modelName"))

models

## End(Not run)
```

`list_models` *List Models*

Description

Returns a list of models from Model Manager

Usage

```
list_models(
  session,
  start = 0,
  limit = 10,
  filters = list(),
  exact = FALSE,
  ...
)
```

Arguments

`session`	viya_connection object, obtained through `session` function
`start`	the index of the first model to return
`limit`	maximum number of models to return
`filters`	list of of names vectors for filter parameters (createdBy, modifiedBy, name). By default it will use the `contains` operation. For more info visit `https://developer.sas.com/apis/rest`
`exact`	boolean, If the filter query should use "contains" for partial match or "eq" for exact match
`...`	additional parameters to be passed to `httr::GET` such as `httr::add_headers`

Value

A `data.frame` with the list of models

Examples

```
## Not run:
models <- list_models(sess, filter = list(createdBy = "creatorUser", name = "modelName"))
models

## End(Not run)
```

`list_modules` *Get list of available models*

Description

Return a `data.frame` of metadata of available models/decisions

Usage

```
list_modules(
  session,
  filters = list(),
  start = 0,
  limit = 20,
  verbose = FALSE,
  exact = FALSE,
  ...
)
```

Arguments

`session`	viya_connection object, obtained through `session` function
`filters`	list of of names vectors for filter parameters (createdBy, modifiedBy, name). By default it will use the `contains` operation. For more info visit `https://developer.sas.com/apis/rest`
`start`	the index of the first module to return
`limit`	maximum number of modules to return
`verbose`	logical, return print API call information
`exact`	boolean, If the filter query should use "contains" for partial match or "eq" for exact match
`...`	additional parameters to be passed to `httr::GET` such as `httr::add_headers`

Value

A `data.frame` with the list of models

Examples

```
## Not run:
models <- list_modules(sess, filters = list(createdBy = 'myUser'))
models

## End(Not run)
```

`list_projects` *List Projects*

Description

Returns a list of projects from Model Manager

Usage

```
list_projects(
  session,
  start = 0,
  limit = 10,
  filters = list(),
  exact = FALSE,
  ...
)
```

Arguments

`session`	viya_connection object, obtained through `session` function
`start`	the index of the first project to return
`limit`	maximum number of projects to return
`filters`	list of of names vectors for filter parameters (createdBy, modifiedBy, name). By default it will use the `contains` operation. For more info visit `https://developer.sas.com/apis/rest`
`exact`	boolean, If the filter query should use "contains" for partial match or "eq" for exact match
`...`	additional parameters to be passed to `httr::GET` such as `httr::add_headers`

Value

A `data.frame` with the list of projects

Examples

```
## Not run:
projects <- list_projects(sess, filter = list(createdBy = "creatorUser", name = "projectName"))
projects

## End(Not run)
```

`list_repositories` *List Repositories*

Description

Returns a list of model repositories. This is required to be able to create model projects

Usage

```
list_repositories(
  session,
  start = 0,
  limit = 10,
  filters = list(),
  exact = FALSE,
  ...
)
```

Arguments

`session`	viya_connection object, obtained through `session` function
`start`	the index of the first project to return
`limit`	maximum number of repositories to return
`filters`	list of of names vectors for filter parameters (createdBy, modifiedBy, name). By default it will use the `contains` operation. For more info visit `https://developer.sas.com/apis/rest`
`exact`	boolean, If the filter query should use "contains" for partial match or "eq" for exact match
`...`	additional parameters to be passed to `httr::GET` such as `httr::add_headers`

Value

A `data.frame` with the list of repositories

Examples

```
## Not run:
repositories <- list_repositories(sess, filter =
                                            list(createdBy = "creatorUser",
                                                 name = "projectName")
                                                 )
repositories

## End(Not run)
```

masScore *Predict MASmodule*

Description

score a data.frame MASmodule/model

Usage

```
masScore(
  session,
  module,
  data,
  exact = TRUE,
  ScoreType = "score",
  forceTrail = TRUE,
  ...
)
```

Arguments

session	viya_connection object, obtained through session function
module	MASmodule object, module name. If name, will try to find a single module with exact name match. See exact parameter
data	data.frame data.frame of the data
exact	the filter query should use "contains" for partial match or "eq" for exact match
ScoreType	execute for decision, score for model
forceTrail	boolean, if the mas model is a decision (execute), it will force add a trailing underscore (_) in variable names.
...	additional parameters to be passed to httr::GET such as httr::add_headers

Details

When the furrr package is installed, masScore uses furrr::future_map_dfr() for scoring, which allows parallel execution via a future plan. Without furrr, scoring runs sequentially using base R. Install both furrr and future to enable parallel scoring.

Value

data.frame with scored rows

Examples

```
## Not run:
# single row (sequential)
scored <- masScore(sess, "module_name", data[1,])
scored

# Parallel scoring — requires the furrr and future packages
# Not recommended for single rows due to parallelisation overhead
future::plan(future::multisession, workers = future::availableCores() - 1)
scored <- masScore(sess, "module_name", data)

# Return to sequential execution
future::plan(future::sequential)

## End(Not run)
```

`model_exists` *Test if a model exists*

Description

Test if the model exists inside SAS Model Manager

Usage

```
model_exists(session, model, ...)
```

Arguments

`session`	viya_connection object, obtained through `session` function
`model`	`MMmodel` object, model ID or model name.
`...`	additional parameters to be passed to `httr::GET` such as `httr::add_headers`

Value

boolean

Examples

```
## Not run:
model_exists(sess, model = ModelProj)

## End(Not run)
```

oauth_consul *Viya oauth token*

Description

Requests a viya oauthtoken

Usage

```
oauth_consul(hostname, consul_token, verbose = FALSE)
```

Arguments

hostname	string, SAS Viya url
consul_token	consul token uuid
verbose	logical, return print API call information

Value

list of API call request data

See Also

Other authentication: refresh_session(), session()

Examples

```
## Not run:
token <- oauth_consul("http://myserver.com",
                consul_token = "47817a5a-3751-4fad-9558-b12b8a702b69") #token is an uuid


## End(Not run)
```

project_exists *Test if a project exists*

Description

Test if the project exists inside SAS Model Manager

Usage

```
project_exists(session, project, ...)
```

Arguments

session	viya_connection object, obtained through session function
project	MMproject object, project ID or project name.
...	additional parameters to be passed to httr::GET such as httr::add_headers

Value

boolean

Examples

```
## Not run:
project_exists(sess, project = ModelProj)

## End(Not run)
```

---

publish_model *Publish Model*

Description

Publish a model from Model Manager to a given destination

Usage

```
publish_model(
  session,
  model,
  name,
  destination = "maslocal",
  exact = TRUE,
  force = FALSE,
  publishInfo = FALSE,
  ...
)
```

Arguments

session	viya_connection object, obtained through session function
model	MMmodel object, project ID or model name. If name, will try to find a single model with exact name match. See exact parameter
name	publish endpoint, if missing, the model name will be used
destination	the publish destination
exact	the filter query should use "contains" for partial match or "eq" for exact match

force	force replace of a published model with the same name
publishInfo	boolean, returns the publishInfo object instead of MASmodule object
...	additional parameters to be passed to httr::POST such as httr::add_headers

Value

A MASmodule object from get_masmodule(). If return_publish_info = TRUE or destination != maslocal, returns a publishInfo object

Examples

```
## Not run:
mas_module <- publish_model(sess, mod, "maslocal", "R_model_published")

scored <- masScore(sess, mas_module, hmeq[1,-1])

## End(Not run)
```

---

refresh_session *Refresh a Viya Session token*

Description

Refresh a viya session object Oauth token

Usage

```
refresh_session(session, verbose = FALSE)
```

Arguments

session	viya_connection object, obtained through session function
verbose	logical, return print API call information

See Also

Other authentication: oauth_consul(), session()

Examples

```
## Not run:
sess <- refresh_session(sess)

## End(Not run)
```

`register_client` *Create a new client*

Description

Create a new client for API call

Usage

```
register_client(
  session,
  client_id,
  client_secret,
  scope = list("openid"),
  access_token_validity = 36000,
  authorized_grant_types = list("client_credentials"),
  authorities = list("uaa.none"),
  additional_parameters = NULL,
  ...
)
```

Arguments

`session`	viya_connection object, obtained through `session` function
`client_id`	name of the new client to be created
`client_secret`	client secret of the new client
`scope`	The scopes allowed for the client to obtain on behalf of users, when using any grant type other than "client_credentials". Groups are treated as scopes. Therefore, the scopes that can be obtained by the client on behalf of a user will be the intersection of the user's groups and the scopes registered to the client via this property. Use the wildcard "" to match all groups. Since SAS Viya allows authorization rules to explicitly deny access to specific groups, SAS recommends always using "". The wildcard "*" will not match internal UAA scopes. This list should always include the scope "openid", which is used to assert the identity of the user that the client is acting on behalf of. For clients that only use the grant type "client_credentials" and therefore do not act on behalf of users, use the default scope "uaa.none".
`access_token_validity`	The time in seconds to access token expiration after it is issued.
`authorized_grant_types`	The list of grant types that can be used to obtain a token with this client. Types can include authorization_code, password, implicit, and client_credentials.
`authorities`	The scopes that the client is able to grant itself when using the "client_credentials" grant type. Wildcards are not allowed.
`additional_parameters`	`list` of parameters and lists to be added to the client creation payload
`...`	additional parameters to be passed to `httr::POST` such as `httr::add_headers`

Value

A sasClient object list

Examples

```
## Not run:
new_client <- register_client(sess, 'my_client', 'my_s3cr3t!')
new_client

## End(Not run)
```

`register_model` *Register a zip file inside model manager*

Description

Registers a zip formatted model in SAS Model Manager.

Usage

```
register_model(
  session,
  file,
  name,
  project,
  type,
  force_pmml_translation = FALSE,
  exact = TRUE,
  force = FALSE,
  model_function = NULL,
  additional_project_parameters = NULL,
  version = "latest",
  project_description = "R SASctl automatic project",
  ...
)
```

Arguments

session	viya_connection object, obtained through session function
file	path to file to be uploaded
name	model name that will be used when registering
project	MMproject object, project ID or project name. If name, will try to find a single project with exact name match. See exact parameter
type	string, pmml, spk, zip or astore

force_pmml_translation
: default is FALSE, set to false will upload pmml as is, but may not work properly. Only if type = "pmml"

exact
: the filter query should use "contains" for partial match or "eq" for exact match

force
: Boolean, force the creation of project if unavailable

model_function
: create_project() parameter of project model function of the created project if force = TRUE. Valid values: analytical, classification, cluster, forecasting, prediction, Text categorization, Text extraction, Text sentiment, Text topics, transformation

additional_project_parameters
: list of additional parameters to be passed to create_project() additional_parameters parameter

version
: This parameter indicates to create a new project version, use the latest version, or use an existing version to import the model into. Valid values are 'NEW', 'LATEST', or a number.

project_description
: description string of additional parameters to be passed to create_project() description parameter

...
: pass to sasctl::vPOST() function

Value

a MMmodel class list

Examples

```
## Not run:
### Building and registering a pmml model

library("pmml")

hmeq <- read.csv("https://support.sas.com/documentation/onlinedoc/viya/exampledatasets/hmeq.csv",
                 stringsAsFactors = TRUE)

hmeq <- na.omit(hmeq)

model1 <- lm(BAD ~ ., hmeq)


saveXML(pmml(model1, model.name="General_Regression_Model",
             app.name="Rattle/PMML",
             description="Linear Regression Model"),
             "my_model.pmml")

output <- register_model(session = sess,
                         file = "my_model.pmml",
                         name = "R_LinearModel",
                         type = "pmml",
                          ## Project UUID example
```

```
                        projectId = "2322da44-9b24-43f6-96f4-456456231")

output

### Bulding and registering an astore model with SWAT

library("swat")

conn <- swat::CAS(hostname = "https://my.sas.server", ## change if needed
            port = 8777,
            username = "sasuser",
            password = "!s3cr3t")

swat::loadActionSet(conn, "astore")
swat::loadActionSet(conn, "decisionTree")


hmeq <- read.csv("https://support.sas.com/documentation/onlinedoc/viya/exampledatasets/hmeq.csv")
castbl <- cas.upload.frame(conn, hmeq)

colinfo <- cas.table.columnInfo(conn, table = castbl)$ColumnInfo
target <- colinfo$Column[1]
inputs <- colinfo$Column[-1]
nominals <- c(target, subset(colinfo, Type == 'varchar')$Column)

dt <- cas.decisionTree.dtreeTrain(conn,
                                  table = castbl,
                                  target = target,
                                  inputs = inputs,
                                  nominals = nominals,
                                  varImp = TRUE,
                                  ## save astore
                                  saveState = list(name = "dt_model_astore",
                                                   replace = TRUE),
                                  casOut = list(name = 'dt_model',
                                                replace = TRUE)
)
dt

## downloading astore
astore_blob <- cas.astore.download(conn,
                                   rstore =  list(name = "dt_model_astore")
)

## saving astore as binary file
astore_path <- "./rf_model.astore"
con <- file(astore_path, "wb")
### file is downloaded as base64 encoded
writeBin(object = jsonlite::base64_dec(astore_blob$blob$data),
         con = con, useBytes = T)

close(con)
```

```
### sasctl connecting
sess <- session(hostname = "https://my.sas.server",
                username = "sasuser",
                password = "!s3cr3t")

output <- register_model(session = sess,
                         file = astore_path,
                         name = "R_swatModel",
                         type = "astore",
                         projectId = "a0c2923b-67e9-4e7f-b5d0-549a04103523")

### Registering a Zip model

output <- register_model(session = sess,
                        file = "model.zip",
                        name = "R_LinearModel",
                        type = "zip",
                        projectId = "2322da44-9b24-43f6-96f4-456456231")

output

## End(Not run)
```

`session` *Viya Session*

Description

Creates a Viya session object to be used in other calls

Usage

```
session(
  hostname,
  username = NULL,
  password = NULL,
  client_id = NULL,
  client_secret = NULL,
  oauth_token = NULL,
  authinfo = NULL,
  auth_code = FALSE,
  verbose = FALSE,
  verify_ssl = TRUE,
  cacert = NULL,
  openBrowser = TRUE,
  platform = TRUE
)
```

Arguments

hostname	string, SAS Viya url
username	string, username for login
password	string, username password
client_id	string, client_id used for authentication, if left blank will use default
client_secret	string, client_secret used for authentication, if left blank will use default
oauth_token	string, if Oauth token is provided, a viya_connection is created
authinfo	A character string that specifies an alternative path to a .authinfo file that is used for authentication. By default, ~/.authinfo is used on Linux and %HOMEDRIVE% or \%HOMEPATH%_authinfo is used on Windows.
auth_code	logical, if TRUE will open a browser with the user and request the authentication code to continue the authentication process. Viya 2022+ only.
verbose	logical, return print API call information
verify_ssl	boolean, verify SSL (Use it with caution)
cacert	ca certificate list
openBrowser	boolean, if TRUE the browser be opened automatically when auth_code = TRUE
platform	logical, make a get call to get platform information (release, OS, siteName)

Value

viya_connection class object

See Also

Other authentication: oauth_consul(), refresh_session()

Examples

```
## Not run:
sess <- session(hostname = "http://myserver.com",
         username = "myuser",
         password = "mysecret")

## End(Not run)
```

update_model *Update a model*

Description

Returns a sasctl MMmodel object from Model Manager

Usage

```
update_model(
  session,
  name,
  model,
  input_vars = NULL,
  output_vars = NULL,
  additional_parameters = NULL,
  update_variables = TRUE,
  exact = TRUE,
  ...
)
```

Arguments

session	viya_connection object, obtained through session function
name	The name of the model
model	MMmodel object, model ID or model name. If name, will try to find a single model with exact name match. See exact parameter
input_vars	data.frame with the input data sample to configure the model variables.
output_vars	data.frame with the output data sample to configure the model variables.
additional_parameters	list of parameters and lists to be added to the payload
update_variables	logical, TRUE, will make additional rest call to update variables using update_model_variables(), variables can't be changed in the same endpoint as other resources
exact	the filter query should use "contains" for partial match or "eq" for exact match
...	additional parameters to be passed to httr::POST such as httr::add_headers

Value

A MMmodel object with the model information

Examples

```
## Not run:
updated_model <- update_model(sess, model = new_model,
                              additional_parameters =
                               list(description = "Updated fancy description",
                              modeler = "BAD"))
updated_model

## End(Not run)
```

`update_model_variables`

Update model variables

Description

Update a SAS Model Manager model variables

Usage

```
update_model_variables(
  session,
  model,
  input_vars = NULL,
  output_vars = NULL,
  exact = TRUE,
  ...
)
```

Arguments

`session`	viya_connection object, obtained through `session` function
`model`	`MMmodel` object, model ID or model name. If name, will try to find a single model with exact name match. See `exact` parameter
`input_vars`	`data.frame` with the input data sample to configure the model variables.
`output_vars`	`data.frame` with the output data sample to configure the model variables.
`exact`	the filter query should use "contains" for partial match or "eq" for exact match
`...`	additional parameters to be passed to `httr::POST` such as `httr::add_headers`

Value

A `MMmodelVariables` variables

Examples

```
## Not run:
new_variables <- update_model_variables(sess, model = new_model,
                                        input_vars = iris[,1:4],
                                        output_vars = iris[,5, drop = F]
                                )
new_variables

## End(Not run)
```

`update_project` *Update a project*

Description

Returns a sasctl MMproject object from Model Manager

Usage

```
update_project(
  session,
  name,
  project,
  input_vars = NULL,
  output_vars = NULL,
  additional_parameters = NULL,
  update_variables = TRUE,
  exact = TRUE,
  ...
)
```

Arguments

`session`	viya_connection object, obtained through `session` function
`name`	The name of the project
`project`	`MMproject` object, project ID or project name. If name, will try to find a single project with exact name match. See `exact` parameter
`input_vars`	`data.frame` with the input data sample to configure the project variables.
`output_vars`	`data.frame` with the output data sample to configure the project variables.
`additional_parameters`	`list` of parameters and lists to be added to the payload
`update_variables`	logical, TRUE, will make additional rest call to update variables using `update_project_variables()`, variables can't be changed in the same endpoint as other resources
`exact`	the filter query should use "contains" for partial match or "eq" for exact match
`...`	additional parameters to be passed to `httr::POST` such as `httr::add_headers`

Value

A data.frame with the list of projects

Examples

```
## Not run:
new_project <- create_project(sess, name = "ModelProj",
                              description = "My fancy project",
                              model_function = "classification")

updated_project <- update_project(sess, project = new_project,
                                  additional_parameters =
                                  list(description = "Updated fancy description",
                                       scoreInputTable = "myTable",
                                       predictionVariable = "BAD"))
updated_project

## End(Not run)
```

---

update_project_variables

Update project variables

Description

Returns a sasctl MMproject object from Model Manager

Usage

```
update_project_variables(
  session,
  project,
  input_vars = NULL,
  output_vars = NULL,
  exact = TRUE,
  sasctl_vars,
  ...
)
```

Arguments

session	viya_connection object, obtained through session function
project	MMproject object, project ID or project name. If name, will try to find a single project with exact name match. See exact parameter
input_vars	data.frame with the input data sample to configure the project variables.
output_vars	data.frame with the output data sample to configure the project variables.

exact	the filter query should use "contains" for partial match or "eq" for exact match
sasctl_vars	data.frame from MMmodel object inputVariables, outputVariables.
...	additional parameters to be passed to httr::POST such as httr::add_headers

Value

A data.frame with the list of projects

Examples

```
## Not run:
new_project <- create_project(sess, name = "ModelProj",
                                    description = "My fancy project",
                                    model_function = "classification")

new_variables <- update_project_variables(sess, project = new_project,
                                          input_vars = iris[,1:4],
                                          output_vars = iris[,5, drop = F])

new_variables

## End(Not run)
```

---

vDELETE *Viya DELETE*

Description

Wrapper to make generic DELETE calls to SAS Viya

Usage

```
vDELETE(session, path, ..., verbose = FALSE)
```

Arguments

session	viya_connection object, obtained through session function
path	character, path to the GET api endpoint
...	additional parameters to be passed to httr::DELETE such as httr::add_headers
verbose	logical, return print API call information

Value

An httr::response() type object

See Also

`httr::GET()`, `httr::POST()`, `httr::PUT()`, `httr::DELETE()`

Other core API requests: `vGET()`, `vHEAD()`, `vPOST()`, `vPUT()`

Examples

```
## Not run:
folders <- vGET(session,
                path = "folders/folders/")


newFolder <- vPOST(session,
                   path = paste0("folders/folders/"),
                   query = list(parentFolderUri = folders$items$parentFolderUri[1]),
                   payload = list(name = "newFolder"))


deletedFolder <- vDELETE(session,
                         path = "folders/folders/",
                         resourceID = newFolder$id)


## End(Not run)
```

---

vGET *Viya GET*

Description

Wrapper to make generic GET calls to SAS Viya

Usage

```
vGET(session, path, ..., query, verbose = FALSE, output = "json")
```

Arguments

session	viya_connection object, obtained through session function
path	character, path to the GET api endpoint
...	additional parameters to be passed to httr::GET such as httr::add_headers
query	list, additional URL query parameters
verbose	logical, return print API call information
output	string, if json will return httr::fromJSON(httr::content(response, as = "text")) with etag information parsed, else if response will return a httr::response() object, if text will return content(response, as = "text")

Value

list if output = "json" default. httr::response() if output = response.

See Also

httr::GET(), httr::POST(), httr::PUT(), httr::DELETE()

Other core API requests: vDELETE(), vHEAD(), vPOST(), vPUT()

Examples

```
## Not run:
folders <- vGET(session,
                path = "folders/folders/")


newFolder <- vPOST(session,
                   path = paste0("folders/folders/"),
                   query = list(parentFolderUri = folders$items$parentFolderUri[1]),
                   payload = list(name = "newFolder"),
                   httr::content_type("application/json"))


deletedFolder <- vDELETE(session,
                         path = "folders/folders/",
                         resourceID = newFolder$id)


## End(Not run)
```

---

vHEAD *Viya HEAD*

Description

Wrapper to make generic DELETE calls to SAS Viya

Usage

```
vHEAD(session, path, payload, ..., verbose = FALSE, output = "response")
```

Arguments

session	viya_connection object, obtained through session function
path	character, path to the GET api endpoint
payload	list or json string, if it is a list, will be transformed in a json string using jsonlite::toJSON
...	additional parameters to be passed to httr::PUT such as httr::add_headers
verbose	logical, return print API call information

output string, if json will return httr::fromJSON(httr::content(response, as = "text")) with etag information parsed, else if response will return a httr::response() object, if text will return content(response, as = "text")

Details

This function in built on top of httr for convenience when calling SAS Viya API endpoints..

Value

list if output = "json" default. httr::response() if output = response.

See Also

httr::GET(), httr::POST(), httr::PUT(), httr::DELETE(), httr::response()

Other core API requests: vDELETE(), vGET(), vPOST(), vPUT()

Examples

```
## Not run:
folders <- vGET(session,
                path = "folders/folders/")


newFolder <- vPOST(session,
                   path = paste0("folders/folders/"),
                   query = list(parentFolderUri = folders$items$parentFolderUri[1]),
                   payload = list(name = "newFolder"),
                   httr::content_type("application/json"))


deletedFolder <- vDELETE(session,
                         path = "folders/folders/",
                         resourceID = newFolder$id)


## End(Not run)
```

---

vPOST *Viya POST*

Description

Wrapper to make generic POST calls to SAS Viya

Usage

```
vPOST(
  session,
  path,
  payload,
  ...,
  query,
  fragment,
  encode = "json",
  verbose = FALSE,
  output = "json"
)
```

Arguments

session	viya_connection object, obtained through session function
path	character, path to the GET api endpoint
payload	list or json string, if it is a list, will be transformed in a json string using jsonlite::toJSON
...	additional parameters to be passed to httr::POST such as httr::add_headers
query	list, additional URL query parameters
fragment	string, additional URL fragment parameter url.com?query#fragment
encode	payload encoding type, to be passed to httr::POST.
verbose	logical, return print API call information
output	string, if json will return httr::fromJSON(httr::content(response, as = "text")) with etag information parsed, else if response will return a httr::response() object, if text will return content(response, as = "text")

Value

list if output = "json" default. httr::response() if output = response.

See Also

httr::GET(), httr::POST(), httr::PUT(), httr::DELETE()

Other core API requests: vDELETE(), vGET(), vHEAD(), vPUT()

Examples

```
## Not run:
folders <- vGET(session,
                path = "folders/folders/")


newFolder <- vPOST(session,
                   path = paste0("folders/folders/"),
                   query = list(parentFolderUri = folders$items$parentFolderUri[1]),
                   payload = list(name = "newFolder"),
```

```
                    httr::content_type("application/json"))


deletedFolder <- vDELETE(session,
                         path = "folders/folders/",
                         resourceID = newFolder$id)


## End(Not run)
```

---

vPUT *Viya PUT*

Description

Wrapper to make generic DELETE calls to SAS Viya

Usage

```
vPUT(
  session,
  path,
  payload,
  ...,
  verbose = FALSE,
  encode = "json",
  output = "json"
)
```

Arguments

session	viya_connection object, obtained through session function
path	character, path to the GET api endpoint
payload	list or json string, if it is a list, will be transformed in a json string using jsonlite::toJSON
...	additional parameters to be passed to httr::PUT such as httr::add_headers
verbose	logical, return print API call information
encode	payload encoding type, to be passed to httr::POST.
output	string, if json will return httr::fromJSON(httr::content(response, as = "text")) with etag information parsed, else if response will return a httr::response() object, if text will return content(response, as = "text")

Details

This function in built on top of httr for convenience when calling SAS Viya API endpoints..

Value

list if `output = "json"` default. `httr::response()` if `output = response`.

See Also

`httr::GET()`, `httr::POST()`, `httr::PUT()`, `httr::DELETE()`, `httr::response()`

Other core API requests: `vDELETE()`, `vGET()`, `vHEAD()`, `vPOST()`

Examples

```
## Not run:
folders <- vGET(session,
                path = "folders/folders/")


newFolder <- vPOST(session,
                   path = paste0("folders/folders/"),
                   query = list(parentFolderUri = folders$items$parentFolderUri[1]),
                   payload = list(name = "newFolder"),
                   httr::content_type("application/json"))


deletedFolder <- vDELETE(session,
                         path = "folders/folders/",
                         resourceID = newFolder$id)


## End(Not run)
```

---

write_fileMetadata_json

Write fileMetadata json

Description

Writes a variable descriptor JSON file for fileMetadata, it will configure the models files metadata within Model Manager in the first upload

Usage

```
write_fileMetadata_json(
  scoreCodeName = "scoreCode.R",
  scoreResource = "model.rda",
  additionalFilesNames = c(),
  additionalFilesRoles = c(),
  path = "./",
  noFile = FALSE
)
```

Arguments

scoreCodeName	Name of the scoring code file
scoreResource	rda file name or other score resources.
additionalFilesNames	additional files names.
additionalFilesRoles	additional files role names.
path	filepath where to write the json (don't include the filename)
noFile	if you don't want to write to a file, only list the output

Value

- list of the mapped properties and values.
- 'ModelProperties.json' file written to path

Examples

```
## Using default names and files
write_fileMetadata_json(noFile = TRUE)

## addition file resources
## send 2 vectors with file names and role name, must be of same length

write_fileMetadata_json(additionalFilesNames = c("myFileName.ext", "myFileName2.ext"),
                        additionalFilesRoles = c("scoreResource", "scoreResource"),
                        noFile = TRUE
                        )
```

---

write_in_out_json *Write variable json*

Description

Writes a variable descriptor JSON file for input or output variables, based on an input dataframe containing predictor and prediction columns.

Usage

```
write_in_out_json(data, input = TRUE, path = "./", noFile = FALSE)
```

Arguments

data	data.frame to map the correct variable types to Json
input	TRUE to write inputVar.json and FALSE to write outputVar.json
path	default to current work dir
noFile	if you don't want to write to a file, only the output

Value

- `list` of the mapped types and sizes.
- 'inputVar.json' or 'outputVar.json' file written to `path`

Examples

```
write_in_out_json(iris[,5, drop = FALSE], input = FALSE, noFile = TRUE)
write_in_out_json(iris[,1:4], input = TRUE, noFile = TRUE)
```

---

write_ModelProperties_json

Write ModelProperties json

Description

Writes a descriptor JSON file for ModelProperties, it will configure the model properties within Model Manager

Usage

```
write_ModelProperties_json(
  modelName,
  modelDescription = "R model",
  modelFunction,
  trainTable = " ",
  algorithm,
  numTargetCategories,
  targetEvent,
  targetVariable,
  eventProbVar,
  modeler = " ",
  tool = "R",
  toolVersion = "default",
  path = "./",
  noFile = FALSE
)
```

Arguments

`modelName`	Name of the model
`modelDescription`	String describing the model
`modelFunction`	Classification, Prediction, Segmentation, Analytical or Clustering.
`trainTable`	Name of the training table
`algorithm`	Algorithm name (Random Forest, GLM, Linear Regression, etc.)

numTargetCategories	number of possible classes for classification
targetEvent	Target event label eg: "1", "versicolor" etc.
targetVariable	Target variable name
eventProbVar	Variable name that has the targetEvent probability Chance
modeler	Modeler's name
tool	Name of the tool used to build the model
toolVersion	Version of the tool used to build the model
path	file path where to write the json (don't include the filename)
noFile	if you don't want to write to a file, only list the output

Value

- list of the mapped properties and values.
- 'ModelProperties.json' file written to path

Examples

```
write_ModelProperties_json(modelName = "My R Model",
                           modelDescription = "Awesome Description",
                           modelFunction = "Classification",
                           trainTable = " ",
                           algorithm = "Logistic Regression",
                           numTargetCategories = 2,
                           targetEvent = "BAD",
                           targetVariable = "P_BAD1",
                           eventProbVar = "P_BAD1",
                           modeler = "John SAS",
                           noFile = TRUE)
```

Index