Package 'AzureML'

February 1, 2016

Type Package

Title Interface with Azure Machine Learning Datasets, Experiments and Web Services

Description Functions and datasets to support Azure Machine Learning. This allows you to interact with datasets, as well as publish and consume R functions as API services.

Version 0.2.10

Date 2016-01-30

Copyright COPYRIGHTS

License MIT + file LICENSE

URL https://github.com/RevolutionAnalytics/AzureML

BugReports https://github.com/RevolutionAnalytics/AzureML/issues

LazyData TRUE

VignetteBuilder knitr

SystemRequirements Requires external zip utility, available in path. On windows, it's sufficient to install RTools.

Imports jsonlite(>= 0.9.16), curl(>= 0.8), foreign, codetools, base64enc, miniCRAN, uuid

Suggests testthat, knitr, rmarkdown, lme4, gbm, MASS

RoxygenNote 5.0.1

NeedsCompilation no

Author Andrie de Vries [aut, cre],
Microsoft Corporation [cph],
Revolution Analytics [cph] (Code adapted from the foreach package)

Maintainer Andrie de Vries <adevries@microsoft.com>

Repository CRAN

Date/Publication 2016-02-01 22:07:52

R topics documented:

`AzureML-package` *Interface to Azure ML Studio datasets and experiments.*

Description

Allows you to work with Azure ML Studio datasets and experiments directly from R.

Summary of functions

1. Create a reference to an Azure ML workspace
   - Workspace: `workspace`
2. Datasets
   - List available datasets: `datasets`
   - Download datasets: `download.datasets`
   - Upload a dataset: `upload.dataset`
   - Delete datasets: `delete.datasets`
3. Experiments

- Get experiments: `experiments`
- Get data from an experiment port: `download.intermediate.dataset`

4. Web Services

- List available services: `services`
- Consume a web service (run data through it and retrieve result): `consume`
- Publish an R function as a web service: `publishWebService`
- Update an existing web service: `updateWebService`
- List web service endpoints: `endpoints`

5. Configure a settings file with your AzureML secrets

The `workspace` function optionally reads your AzureML credentials from a settings file located at `~/.azureml/settings.json`. You can read and write this file using:

- Write: `write.AzureML.config`
- Read: `read.AzureML.config`

`consume` *Use a web service to score data in list (key=value) format*

Description

Score data represented as lists where each list key represents a parameter of the web service.

Usage

```
consume(endpoint, ..., globalParam, retryDelay = 10, output = "output1")
```

Arguments

`endpoint`	Either an AzureML web service endpoint returned by `publishWebService`, `endpoints`, or simply an AzureML web service from `services`; in the latter case the default endpoint for the service will be used.
`...`	variable number of requests entered as lists in key-value format; optionally a single data frame argument.
`globalParam`	global parameters entered as a list, default value is an empty list
`retryDelay`	the time in seconds to delay before retrying in case of a server error
`output`	name of the output port to return usually 'output1' or 'output2'; set to NULL to return everything as raw results in JSON-encoded list form

Value

data frame containing results returned from web service call

Note

Set ... to a list of key/value pairs corresponding to web service inputs. Optionally, set ... to a single data frame with columns corresponding to web service variables. The data frame approach returns output from the evaluation of each row of the data frame (see the examples).

See Also

publishWebService endpoints services workspace

Other consumption functions: workspace

Examples

```
## Not run:
# Use a default configuration in ~/.azureml, alternatively
# see help for `?workspace`.

ws <- workspace()

# Publish a simple model using the lme4::sleepdata ----------------------------

library(lme4)
set.seed(1)
train <- sleepstudy[sample(nrow(sleepstudy), 120),]
m <- lm(Reaction ~ Days + Subject, data = train)

# Deine a prediction function to publish based on the model:
sleepyPredict <- function(newdata){
  predict(m, newdata=newdata)
}

ep <- publishWebService(ws, fun = sleepyPredict, name="sleepy lm",
                        inputSchema = sleepstudy,
                        data.frame=TRUE)

# OK, try this out, and compare with raw data
ans <- consume(ep, sleepstudy)$ans
plot(ans, sleepstudy$Reaction)

# Remove the service
deleteWebService(ws, "sleepy lm")



# Another data frame example --------------------------------------------------

# If your function can consume a whole data frame at once, you can also
# supply data in that form, resulting in more efficient computation.
# The following example builds a simple linear model on a subset of the
# airquality data and publishes a prediction function based on the model.
set.seed(1)
m <- lm(Ozone ~ ., data=airquality[sample(nrow(airquality), 100),])
```

```
# Define a prediction function based on the model:
fun <- function(newdata)
{
  predict(m, newdata=newdata)
}
# Note the definition of inputSchema and use of the data.frame argument.
ep <- publishWebService(ws, fun=fun, name="Ozone",
                        inputSchema = airquality,
                        data.frame=TRUE)
ans <- consume(ep, airquality)$ans
plot(ans, airquality$Ozone)
deleteWebService(ws, "Ozone")



# Train a model using diamonds in ggplot2 -----------------------------------
# This example also demonstrates how to deal with factor in the data

data(diamonds, package="ggplot2")
set.seed(1)
train_idx = sample.int(nrow(diamonds), 30000)
test_idx = sample(setdiff(seq(1, nrow(diamonds)), train_idx), 500)
train <- diamonds[train_idx, ]
test  <- diamonds[test_idx, ]

model <- glm(price ~ carat + clarity + color + cut - 1, data = train,
             family = Gamma(link = "log"))

diamondLevels <- diamonds[1, ]

# The model works reasonably well, except for some outliers
plot(exp(predict(model, test)) ~ test$price)

# Create a prediction function that converts characters correctly to factors

predictDiamonds <- function(x){
  x$cut     <- factor(x$cut,
                      levels = levels(diamondLevels$cut), ordered = TRUE)
  x$clarity <- factor(x$clarity,
                      levels = levels(diamondLevels$clarity), ordered = TRUE)
  x$color   <- factor(x$color,
                      levels = levels(diamondLevels$color), ordered = TRUE)
  exp(predict(model, newdata = x))
}


# Publish the service

ws <- workspace()
ep <- publishWebService(ws, fun = predictDiamonds, name = "diamonds",
                        inputSchema = test,
                        data.frame = TRUE
)
```

```
# Consume the service
results <- consume(ep, test)$ans
plot(results ~ test$price)

deleteWebService(ws, "diamonds")



# Simple example using scalar input ----------------------------------------

ws <- workspace()

# Really simple example:
add <- function(x,y) x + y
endpoint <- publishWebService(ws,
                              fun = add,
                              name = "addme",
                              inputSchema = list(x="numeric",
                                                 y="numeric"),
                              outputSchema = list(ans="numeric"))
consume(endpoint, list(x=pi, y=2))

# Now remove the web service named "addme" that we just published
deleteWebService(ws, "addme")



# Send a custom R function for evaluation in AzureML -------------------------

# A neat trick to evaluate any expression in the Azure ML virtual
# machine R session and view its output:
ep <- publishWebService(ws,
                        fun =  function(expr) {
                          paste(capture.output(
                            eval(parse(text=expr))), collapse="\n")
                        },
                        name="commander",
                        inputSchema = list(x = "character"),
                        outputSchema = list(ans = "character"))
cat(consume(ep, list(x = "getwd()"))$ans)
cat(consume(ep, list(x = ".packages(all=TRUE)"))$ans)
cat(consume(ep, list(x = "R.Version()"))$ans)

# Remove the service we just published
deleteWebService(ws, "commander")



# Understanding the scoping rules ------------------------------------------

# The following example illustrates scoping rules. Note that the function
# refers to the variable y defined outside the function body. That value
```

```
# will be exported with the service.
y <- pi
ep <- publishWebService(ws,
                        fun = function(x) x + y,
                        name = "lexical scope",
                        inputSchema = list(x = "numeric"),
                        outputSchema = list(ans = "numeric"))
cat(consume(ep, list(x=2))$ans)

# Remove the service we just published
deleteWebService(ws, "lexical scope")


# Demonstrate scalar inputs but sending a data frame for scoring -------------

# Example showing the use of consume to score all the rows of a data frame
# at once, and other invocations for evaluating multiple sets of input
# values. The columns of the data frame correspond to the input parameters
# of the web service in this example:
f <- function(a,b,c,d) list(sum = a+b+c+d, prod = a*b*c*d)
ep <-  publishWebService(ws,
                         f,
                         name = "rowSums",
                         inputSchema = list(
                           a = "numeric",
                           b = "numeric",
                           c = "numeric",
                           d = "numeric"
                         ),
                         outputSchema = list(
                           sum ="numeric",
                           prod = "numeric")
)
x <- head(iris[,1:4])  # First four columns of iris

# Note the following will FAIL because of a name mismatch in the arguments
# (with an informative error):
consume(ep, x, retryDelay=1)
# We need the columns of the data frame to match the inputSchema:
names(x) <- letters[1:4]
# Now we can evaluate all the rows of the data frame in one call:
consume(ep, x)
# output should look like:
#    sum    prod
# 1 10.2   4.998
# 2  9.5   4.116
# 3  9.4  3.9104
# 4  9.4   4.278
# 5 10.2    5.04
# 6 11.4 14.3208

# You can use consume to evaluate just a single set of input values with this
# form:
```

```
consume(ep, a=1, b=2, c=3, d=4)

# or, equivalently,
consume(ep, list(a=1, b=2, c=3, d=4))

# You can evaluate multiple sets of input values with a data frame input:
consume(ep, data.frame(a=1:2, b=3:4, c=5:6, d=7:8))

# or, equivalently, with multiple lists:
consume(ep, list(a=1, b=3, c=5, d=7), list(a=2, b=4, c=6, d=8))

# Remove the service we just published
deleteWebService(ws, "rowSums")

# A more efficient way to do the same thing using data frame input/output:
f <- function(df) with(df, list(sum = a+b+c+d, prod = a*b*c*d))
ep = publishWebService(ws, f, name="rowSums2",
                       inputSchema = data.frame(a = 0, b = 0, c = 0, d = 0))
consume(ep, data.frame(a=1:2, b=3:4, c=5:6, d=7:8))
deleteWebService(ws, "rowSums2")



# Automatically discover dependencies ----------------------------------------

# The publishWebService function uses `miniCRAN` to include dependencies on
# packages required by your function. The next example uses the `lmer`
# function from the lme4 package, and also shows how to publish a function
# that consumes a data frame by setting data.frame=TRUE.  Note! This example
# depends on a lot of packages and may take some time to upload to Azure.
library(lme4)
# Build a sample mixed effects model on just a subset of the sleepstudy data...
set.seed(1)
m <- lmer(Reaction ~ Days + (Days | Subject),
          data=sleepstudy[sample(nrow(sleepstudy), 120),])
# Deine a prediction function to publish based on the model:
fun <- function(newdata)
{
  predict(m, newdata=newdata)
}
ep <- publishWebService(ws, fun=fun, name="sleepy lmer",
                        inputSchema= sleepstudy,
                        packages="lme4",
                        data.frame=TRUE)

# OK, try this out, and compare with raw data
ans = consume(ep, sleepstudy)$ans
plot(ans, sleepstudy$Reaction)

# Remove the service
deleteWebService(ws, "sleepy lmer")

## End(Not run)
```

consumeDataframe *Deprecated functions*

Description

Deprecated functions

Usage

```
consumeDataframe()

consumeFile()

consumeLists()

getEPDetails()

getWSDetails()
```

datasets *List datasets in an AzureML workspace.*

Description

List datasets in an AzureML workspace, optionally filtering on sample or my datasets.

Usage

```
datasets(ws, filter = c("all", "my datasets", "samples"))
```

Arguments

ws	An AzureML workspace reference returned by workspace.
filter	Optionally filter result, returing all, mine, or sample datasets.

Value

A data.frame with class Datasets listing available datasets in the workspace.

Note

datasets(w) is equivalent to w$datasets. Since w$datasets is simply an R data.frame, you can alternatively filter on any variable as desired.

See Also

workspace, experiments, download.datasets

Other dataset functions: delete.datasets; download.intermediate.dataset; upload.dataset; workspace

Examples

```
## Not run:
  library(AzureML)

  # Use the default config file ~/azureml/settings.json with format:
  #   {"workspace":{
  #     "id":"test_id",
  #     "authorization_token": "test_token",
  #     "api_endpoint":"api_endpoint",
  #     "management_endpoint":"management_endpoint"
  #    }}
  # or, optionally set the `id` and `auth` parameters in the workspace
  # function.
  ws <- workspace()

  # List datasets
  ws$datasets
  datasets(ws)

  dataset <- "New York weather"
  ds <- match(dataset, ws$datasets$Name)
  frame <- download.datasets(ws$datasets[ds, ])
  head(frame)

  # Alternative approach:
  frame <- download.datasets(ws, name=dataset)
  head(frame)

## End(Not run)
```

---

delete.datasets *Delete datasets from an AzureML workspace.*

Description

Delete datasets from an AzureML workspace.

Usage

```
delete.datasets(ws, name, host)
```

Arguments

ws	An AzureML workspace reference returned by workspace.
name	Either one or more Dataset objects (rows from the workspace datasets data frame), or a character vector of dataset names to delete.
host	AzureML delete service endpoint

Value

A data frame with columns Name, Deleted, status_code indicating the HTTP status code and success/failure result of the delete operation for each dataset.

See Also

Other dataset functions: datasets; download.intermediate.dataset; upload.dataset; workspace

---

deleteWebService *Delete a Microsoft Azure Web Service*

Description

Delete a Microsoft Azure Machine Learning web service from your workspace.

Usage

```
deleteWebService(ws, name, refresh = TRUE)
```

Arguments

ws	An AzureML workspace reference returned by workspace.
name	Either one row from the workspace services data.frame corresponding to a service to delete, or simply a service name character string.
refresh	Set to FALSE to supress automatic updating of the workspace list of services, useful when deleting many services in bulk.

Value

The updated data.frame of workspace services is invisibly returned.

Note

If more than one service matches the supplied name, the first listed service will be deleted.

See Also

services publishWebService updateWebService

Other publishing functions: publishWebService, updateWebService; workspace

Examples

```
## Not run:
# Use a default configuration in ~/.azureml, alternatively
# see help for `?workspace`.

ws <- workspace()

# Publish a simple model using the lme4::sleepdata ----------------------------

library(lme4)
set.seed(1)
train <- sleepstudy[sample(nrow(sleepstudy), 120),]
m <- lm(Reaction ~ Days + Subject, data = train)

# Deine a prediction function to publish based on the model:
sleepyPredict <- function(newdata){
  predict(m, newdata=newdata)
}

ep <- publishWebService(ws, fun = sleepyPredict, name="sleepy lm",
                        inputSchema = sleepstudy,
                        data.frame=TRUE)

# OK, try this out, and compare with raw data
ans <- consume(ep, sleepstudy)$ans
plot(ans, sleepstudy$Reaction)

# Remove the service
deleteWebService(ws, "sleepy lm")



# Another data frame example --------------------------------------------------

# If your function can consume a whole data frame at once, you can also
# supply data in that form, resulting in more efficient computation.
# The following example builds a simple linear model on a subset of the
# airquality data and publishes a prediction function based on the model.
set.seed(1)
m <- lm(Ozone ~ ., data=airquality[sample(nrow(airquality), 100),])
# Define a prediction function based on the model:
fun <- function(newdata)
{
  predict(m, newdata=newdata)
}
# Note the definition of inputSchema and use of the data.frame argument.
ep <- publishWebService(ws, fun=fun, name="Ozone",
                        inputSchema = airquality,
                        data.frame=TRUE)
ans <- consume(ep, airquality)$ans
plot(ans, airquality$Ozone)
deleteWebService(ws, "Ozone")
```

```
# Train a model using diamonds in ggplot2 -----------------------------------
# This example also demonstrates how to deal with factor in the data

data(diamonds, package="ggplot2")
set.seed(1)
train_idx = sample.int(nrow(diamonds), 30000)
test_idx = sample(setdiff(seq(1, nrow(diamonds)), train_idx), 500)
train <- diamonds[train_idx, ]
test  <- diamonds[test_idx, ]

model <- glm(price ~ carat + clarity + color + cut - 1, data = train,
             family = Gamma(link = "log"))

diamondLevels <- diamonds[1, ]

# The model works reasonably well, except for some outliers
plot(exp(predict(model, test)) ~ test$price)

# Create a prediction function that converts characters correctly to factors

predictDiamonds <- function(x){
  x$cut     <- factor(x$cut,
                      levels = levels(diamondLevels$cut), ordered = TRUE)
  x$clarity <- factor(x$clarity,
                      levels = levels(diamondLevels$clarity), ordered = TRUE)
  x$color   <- factor(x$color,
                      levels = levels(diamondLevels$color), ordered = TRUE)
  exp(predict(model, newdata = x))
}


# Publish the service

ws <- workspace()
ep <- publishWebService(ws, fun = predictDiamonds, name = "diamonds",
                        inputSchema = test,
                        data.frame = TRUE
)

# Consume the service
results <- consume(ep, test)$ans
plot(results ~ test$price)

deleteWebService(ws, "diamonds")



# Simple example using scalar input -----------------------------------------

ws <- workspace()
```

```
# Really simple example:
add <- function(x,y) x + y
endpoint <- publishWebService(ws,
                              fun = add,
                              name = "addme",
                              inputSchema = list(x="numeric",
                                                 y="numeric"),
                              outputSchema = list(ans="numeric"))
consume(endpoint, list(x=pi, y=2))

# Now remove the web service named "addme" that we just published
deleteWebService(ws, "addme")



# Send a custom R function for evaluation in AzureML -------------------------

# A neat trick to evaluate any expression in the Azure ML virtual
# machine R session and view its output:
ep <- publishWebService(ws,
                        fun =  function(expr) {
                          paste(capture.output(
                            eval(parse(text=expr))), collapse="\n")
                        },
                        name="commander",
                        inputSchema = list(x = "character"),
                        outputSchema = list(ans = "character"))
cat(consume(ep, list(x = "getwd()"))$ans)
cat(consume(ep, list(x = ".packages(all=TRUE)"))$ans)
cat(consume(ep, list(x = "R.Version()"))$ans)

# Remove the service we just published
deleteWebService(ws, "commander")



# Understanding the scoping rules --------------------------------------------

# The following example illustrates scoping rules. Note that the function
# refers to the variable y defined outside the function body. That value
# will be exported with the service.
y <- pi
ep <- publishWebService(ws,
                        fun = function(x) x + y,
                        name = "lexical scope",
                        inputSchema = list(x = "numeric"),
                        outputSchema = list(ans = "numeric"))
cat(consume(ep, list(x=2))$ans)

# Remove the service we just published
deleteWebService(ws, "lexical scope")
```

```
# Demonstrate scalar inputs but sending a data frame for scoring -------------

# Example showing the use of consume to score all the rows of a data frame
# at once, and other invocations for evaluating multiple sets of input
# values. The columns of the data frame correspond to the input parameters
# of the web service in this example:
f <- function(a,b,c,d) list(sum = a+b+c+d, prod = a*b*c*d)
ep <-  publishWebService(ws,
                         f,
                         name = "rowSums",
                         inputSchema = list(
                           a = "numeric",
                           b = "numeric",
                           c = "numeric",
                           d = "numeric"
                         ),
                         outputSchema = list(
                           sum ="numeric",
                           prod = "numeric")
)
x <- head(iris[,1:4])  # First four columns of iris

# Note the following will FAIL because of a name mismatch in the arguments
# (with an informative error):
consume(ep, x, retryDelay=1)
# We need the columns of the data frame to match the inputSchema:
names(x) <- letters[1:4]
# Now we can evaluate all the rows of the data frame in one call:
consume(ep, x)
# output should look like:
#    sum    prod
# 1 10.2   4.998
# 2  9.5   4.116
# 3  9.4  3.9104
# 4  9.4   4.278
# 5 10.2    5.04
# 6 11.4 14.3208

# You can use consume to evaluate just a single set of input values with this
# form:
consume(ep, a=1, b=2, c=3, d=4)

# or, equivalently,
consume(ep, list(a=1, b=2, c=3, d=4))

# You can evaluate multiple sets of input values with a data frame input:
consume(ep, data.frame(a=1:2, b=3:4, c=5:6, d=7:8))

# or, equivalently, with multiple lists:
consume(ep, list(a=1, b=3, c=5, d=7), list(a=2, b=4, c=6, d=8))

# Remove the service we just published
```

```
deleteWebService(ws, "rowSums")

# A more efficient way to do the same thing using data frame input/output:
f <- function(df) with(df, list(sum = a+b+c+d, prod = a*b*c*d))
ep = publishWebService(ws, f, name="rowSums2",
                       inputSchema = data.frame(a = 0, b = 0, c = 0, d = 0))
consume(ep, data.frame(a=1:2, b=3:4, c=5:6, d=7:8))
deleteWebService(ws, "rowSums2")



# Automatically discover dependencies ----------------------------------------

# The publishWebService function uses `miniCRAN` to include dependencies on
# packages required by your function. The next example uses the `lmer`
# function from the lme4 package, and also shows how to publish a function
# that consumes a data frame by setting data.frame=TRUE.  Note! This example
# depends on a lot of packages and may take some time to upload to Azure.
library(lme4)
# Build a sample mixed effects model on just a subset of the sleepstudy data...
set.seed(1)
m <- lmer(Reaction ~ Days + (Days | Subject),
          data=sleepstudy[sample(nrow(sleepstudy), 120),])
# Deine a prediction function to publish based on the model:
fun <- function(newdata)
{
  predict(m, newdata=newdata)
}
ep <- publishWebService(ws, fun=fun, name="sleepy lmer",
                        inputSchema= sleepstudy,
                        packages="lme4",
                        data.frame=TRUE)

# OK, try this out, and compare with raw data
ans = consume(ep, sleepstudy)$ans
plot(ans, sleepstudy$Reaction)

# Remove the service
deleteWebService(ws, "sleepy lmer")

## End(Not run)
```

`discoverSchema` *Discover web service schema.*

Description

Discover the expected input to a web service specified by a web service ID ng the workspace ID and web service ID, information specific to the consumption functions

Usage

```
discoverSchema(helpURL, scheme = "https",
  host = "ussouthcentral.services.azureml.net", api_version = "2.0")
```

Arguments

helpURL	URL of the help page of the web service
scheme	the URI scheme
host	optional parameter that defaults to ussouthcentral.services.azureml.net
api_version	AzureML API version

Value

List containing the request URL of the webservice, column names of the data, sample input as well as the input schema

See Also

publishWebService consume workspace link{services} endpoints endpointHelp

Other discovery functions: endpointHelp; endpoints, getEndpoints; getWebServices, services; workspace

---

download.datasets *Download one or more datasets from an AzureML workspace.*

Description

Download one or more datasets from an AzureML workspace into local R data frame or raw binary objects.

Usage

```
download.datasets(dataset, name, ...)
```

Arguments

dataset	Either one or more rows from a datasets data frame in a workspace, or just a workspace from workspace. When source is a workspace, then the name parameter must also be specified.
name	Optional character vector of one or more dataset names to filter the datasets parameter list by.
...	Optional arguments to pass to read.table for CSV or TSV DataTypeIds or to readBin for the ZIP DataTypeId. For example, specify stringsAsFactors=TRUE if you wish, or any other valid argument to read.table.

Value

If one dataset is specified (that is, one row from a workspace `datasets` data frame), then a single data frame is returned. If more than one dataset is specified (more than one row), then a list of data frames is returned.

Note

TSV- and CSV-formatted datasets return data frame results with `stringsAsFactors=FALSE` by default (independently of the global `stringsAsFactors` option).

This function can download datasets with various CSV and TSV "DataTypeIds", or "DataTypeId" of "ARFF", "PlainText" or "ZIP". Other "DataTypeIds" return an error. See the AzureML Data Format Conversion modules to convert data to a supported format. Data with DataTypeId "ZIP" are returned in a raw binary R vector, which could then be passed through `unzip`, for example.

See Also

`workspace`, `datasets`, `read.table`, `download.intermediate.dataset`

Examples

```
## Not run:
  library(AzureML)

  name <- "Blood donation data"

  ws <- workspace()

  # The following three alternatives produce the same output:
  frame1 <- download.datasets(ws, name)
  frame2 <- download.datasets(datasets(ws), name)

  # Note that one can examine all the names, sizes, etc. of the datasets
  # in ws by examining d:
  d <- datasets(ws)
  frame3 <- download.datasets(subset(d, Name == name))

  head(frame1)

## End(Not run)
```

---

download.intermediate.dataset

Download a dataset from an AzureML experiment module.

Description

Allows you to download the data from certain types of modules in AzureML experiments. You can generate the information required from AzureML Studio by (right) clicking on a module output port and selecting the option "Generate Data Access Code...".

Usage

```
download.intermediate.dataset(ws, experiment, node_id,
  port_name = "Results dataset", data_type_id = "GenericCSV", ...)
```

Arguments

ws	An AzureML workspace reference returned by workspace.
experiment	AzureML experiment ID.
node_id	Experiment node ID.
port_name	Experiment port name. The default is "Results dataset".
data_type_id	Experiment data type id. The default is "GenericCSV". See the note below for other types.
...	Optional arguments to pass to read.table for CSV or TSV DataTypeIds. For example, specify stringsAsFactors=TRUE if you wish, or any other valid argument to read.table.

Value

In most cases a data frame. Exceptions are: a raw vector for DataTypeId="Zip" and character vector for DataTypeId="PlainText"

Note

TSV- and CSV-formatted datasets return data frame results with stringsAsFactors=FALSE by default (independently of the global stringsAsFactors option).

Supported DataTypeId options

This function can download datasets with various CSV and TSV DataTypeId (with or without headers), in addition to "ARFF", "PlainText" and "Zip". Other "DataTypeIds" return an error. See the AzureML Data Format Conversion modules to convert data to a supported format.

See Also

workspace, datasets, read.table and download.datasets

Other dataset functions: datasets; delete.datasets; upload.dataset; workspace

Other experiment functions: experiments; workspace

endpointHelp *Display AzureML Web Service Endpoint Help Screens*

Description

Download and return help for the specified AzureML web service endpoint.

Usage

```
endpointHelp(e, type = c("apidocument", "r-snippet", "score", "jobs",
  "update"))
```

Arguments

e	an AzureML web service endpoint from the endpoints function.
type	the type of help to display.

Value

The help text is returned. If type="apidocument", then the help is returned as a list from a parsed JSON document describing the service.

See Also

Other discovery functions: discoverSchema; endpoints, getEndpoints; getWebServices, services; workspace

Examples

```
## Not run:
workspace_id <- ""          # Your AzureML workspace id
authorization_token <- ""   # Your AzureML authorization token

ws <- workspace(
  id = workspace_id,
  auth = authorization_token
)

s <- services(ws)
e <- endpoints(ws, s[1,])
endpointHelp(e[1,])

Particularly useful way to see expected service input and output:
endpointHelp(e[1,])$definitions



## End(Not run)
```

endpoints *List AzureML Web Service Endpoints*

Description

Return a list of web services endpoints for the specified web service id.

Usage

```
endpoints(ws, service_id, endpoint_id, host = ws$.management_endpoint)

getEndpoints(ws, service_id, endpoint_id, host = ws$.management_endpoint)
```

Arguments

ws	An AzureML workspace reference returned by workspace.
service_id	A web service Id, for example returned by services; alternatively a row from the services data frame identifying the service.
endpoint_id	An optional endpoint id. If supplied, return the endpoint information for just that id. Leave undefined to return a data.frame of all end points associated with the service.
host	The AzureML web services URI

Value

Returns a data.frame with variables:

- Name
- Description
- CreationTime
- WorkspaceId
- WebServiceId
- HelpLocation
- PrimaryKey
- SecondaryKey
- ApiLocation
- Version
- MaxConcurrentCalls
- DiagnosticsTraceLevel
- ThrottleLevel

Each row of the data.frame corresponds to an end point.

Note

`getEndPoints` is an alias for `endpoints`.

See Also

Other discovery functions: `discoverSchema`; `endpointHelp`; `getWebServices`, `services`; `workspace`

Examples

```
## Not run:
workspace_id <- ""          # Your AzureML workspace id
authorization_token <- ""   # Your AsureML authorization token

ws <- workspace(
  id = workspace_id,
  auth = authorization_token
)

s <- services(ws)
endpoints(ws, s$Id[1])

# Note that you can alternatively just use the entire row that
# describes the service.
endpoints(ws, s[1,])

# Equivalent:
getEndpoints(ws, s$Id[1])

## End(Not run)
```

`experiments` *List experiments in an AzureML workspace.*

Description

List experiments in an AzureML workspace, optionally filtering on sample or my experiments.

Usage

```
experiments(ws, filter = c("all", "my datasets", "samples"))
```

Arguments

`ws`	An AzureML workspace reference returned by `workspace`.
`filter`	Optionally filter result, returing all, mine, or sample datasets.

Value

A data.frame with class `Experiments` listing available experiments in the workspace.

Note

`experiments(w)` is equivalent to `w$experiments`. Since `w$experiments` is simply an R data.frame, you can alternatively filter on any variable as desired.

See Also

`workspace`, `datasets`, `download.intermediate.dataset`

Other experiment functions: `download.intermediate.dataset`; `workspace`

Examples

```
## Not run:
  library(AzureML)

  experiment <- "dd01c7e4a424432c9a9f83142d5cfec4.f-id.d2f351dd4cec4c06a4592ac83f7af55a"
  node_id <- '2a472ae1-ecb1-4f40-ae4e-cd3cecb1003f-268'

  ws <- workspace()

  ws$experiments
  experiments(ws)
  frame <- download.intermediate.dataset(ws, experiment, node_id,
                                         port_name = "Results dataset",
                                         data_type_id = "GenericCSV")
  head(frame)

## End(Not run)
```

`is.Endpoint` *Test if an object is an Azure ML Endpoint.*

Description

Test if an object is an Azure ML Endpoint.

Usage

```
is.Endpoint(x)
```

Arguments

`x`	an R object

Value

logical value, TRUE if `x` represents an Azure ML web service endpoint

is.Service *Test if an object is an Azure ML Service.*

Description

Test if an object is an Azure ML Service.

Usage

```
is.Service(x)
```

Arguments

x	an R object

Value

logical value, TRUE if x represents an Azure ML web service

is.Workspace *Test if an object is an Azure ML Workspace.*

Description

Test if an object is an Azure ML Workspace.

Usage

```
is.Workspace(x)
```

Arguments

x	an R object

Value

logical value, TRUE if x represents an Azure ML workspace.

publishWebService *Publish a function as a Microsoft Azure Web Service.*

Description

Publish a function to Microsoft Azure Machine Learning as a web service. The web service created is a standard Azure ML web service, and can be used from any web or mobile platform as long as the user knows the API key and URL. The function to be published is limited to inputs/outputs consisting of lists of scalar values or single data frames (see the notes below and examples). Requires a zip program to be installed (see note below).

Usage

```
publishWebService(ws, fun, name, inputSchema, outputSchema,
  data.frame = FALSE, export = character(0), noexport = character(0),
  packages, version = "3.1.0", serviceId, host = ws$.management_endpoint)

updateWebService(ws, fun, name, inputSchema, outputSchema, data.frame = FALSE,
  export = character(0), noexport = character(0), packages,
  version = "3.1.0", serviceId, host = ws$.management_endpoint)
```

Arguments

ws	An AzureML workspace reference returned by workspace.
fun	a function to publish; the function must have at least one argument.
name	name of the new web service; ignored when serviceId is specified (when updating an existing web service).
inputSchema	either a list of fun input parameters and their AzureML types formatted as list("arg1"="type", "arg2"="type", ...), or an example input data frame when fun takes a single data frame argument; see the note below for details.
outputSchema	list of fun outputs and AzureML types, formatted as list("output1"="type", "output2"="type", .. optional when inputSchema is an example input data frame.
data.frame	TRUE indicates that the function fun accepts a data frame as input and returns a data frame output; automatically set to TRUE when inputSchema is a data frame.
export	optional character vector of variable names to explicitly export in the web service for use by the function. See the note below.
noexport	optional character vector of variable names to prevent from exporting in the web service.
packages	optional character vector of R packages to bundle in the web service, including their dependencies.
version	optional R version string for required packages (the version of R running in the AzureML Web Service).
serviceId	optional Azure web service ID; use to update an existing service (see Note below).
host	optional Azure regional host, defaulting to the global management_endpoint set in workspace

Value

A data.frame describing the new service endpoints, cf. endpoints. The output can be directly used by the consume function.

Note

Data Types

AzureML data types are different from, but related to, R types. You may specify the R types numeric, logical, integer, and character and those will be specified as AzureML types double, boolean, int32, string, respectively.

Input and output schemas

Function input must be:

1. named scalar arguments with names and types specified in inputSchema
2. one or more lists of named scalar values
3. a single data frame when data.frame=TRUE is specified; either explicitly specify the column names and types in inputSchema or provide an example input data frame as inputSchema

Function output is always returned as a data frame with column names and types specified in outputSchema. See the examples for example use of all three I/O options.

Updating a web service

Leave the serviceId parameter undefined to create a new AzureML web service, or specify the ID of an existing web service to update it, replacing the function, inputSchema, outputSchema, and required R pacakges with new values. The name parameter is ignored serviceId is specified to update an existing web service.

The updateWebService function is nearly an alias for publishWebService, differing only in that the serviceId parameter is required by updateWebService.

The publishWebService function automatically exports objects required by the function to a working environment in the AzureML machine, including objects accessed within the function using lexical scoping rules. Use the exports parameter to explicitly include other objects that are needed. Use noexport to explicitly prevent objects from being exported.

Note that it takes some time to update the AzureML service on the server. After updating the service, you may have to wait several seconds for the service to update. The time it takes will depend on a number of factors, including the complexity of your web service function.

External zip program required

The function uses zip to compress information before transmission to AzureML. To use this, you need to have a zip program installed on your machine, and this program should be available in the path. The program should be called zip otherwise R may not find it. On windows, it is sufficient to install RTools (see https://cran.r-project.org/bin/windows/Rtools/)

See Also

endpoints, discoverSchema, consume and services.

Other publishing functions: deleteWebService; workspace

Examples

```
## Not run:
# Use a default configuration in ~/.azureml, alternatively
# see help for `?workspace`.

ws <- workspace()

# Publish a simple model using the lme4::sleepdata ----------------------------

library(lme4)
set.seed(1)
train <- sleepstudy[sample(nrow(sleepstudy), 120),]
m <- lm(Reaction ~ Days + Subject, data = train)

# Deine a prediction function to publish based on the model:
sleepyPredict <- function(newdata){
  predict(m, newdata=newdata)
}

ep <- publishWebService(ws, fun = sleepyPredict, name="sleepy lm",
                        inputSchema = sleepstudy,
                        data.frame=TRUE)

# OK, try this out, and compare with raw data
ans <- consume(ep, sleepstudy)$ans
plot(ans, sleepstudy$Reaction)

# Remove the service
deleteWebService(ws, "sleepy lm")



# Another data frame example --------------------------------------------------

# If your function can consume a whole data frame at once, you can also
# supply data in that form, resulting in more efficient computation.
# The following example builds a simple linear model on a subset of the
# airquality data and publishes a prediction function based on the model.
set.seed(1)
m <- lm(Ozone ~ ., data=airquality[sample(nrow(airquality), 100),])
# Define a prediction function based on the model:
fun <- function(newdata)
{
  predict(m, newdata=newdata)
}
# Note the definition of inputSchema and use of the data.frame argument.
ep <- publishWebService(ws, fun=fun, name="Ozone",
                        inputSchema = airquality,
                        data.frame=TRUE)
ans <- consume(ep, airquality)$ans
plot(ans, airquality$Ozone)
deleteWebService(ws, "Ozone")
```

```
# Train a model using diamonds in ggplot2 -----------------------------------
# This example also demonstrates how to deal with factor in the data

data(diamonds, package="ggplot2")
set.seed(1)
train_idx = sample.int(nrow(diamonds), 30000)
test_idx = sample(setdiff(seq(1, nrow(diamonds)), train_idx), 500)
train <- diamonds[train_idx, ]
test  <- diamonds[test_idx, ]

model <- glm(price ~ carat + clarity + color + cut - 1, data = train,
             family = Gamma(link = "log"))

diamondLevels <- diamonds[1, ]

# The model works reasonably well, except for some outliers
plot(exp(predict(model, test)) ~ test$price)

# Create a prediction function that converts characters correctly to factors

predictDiamonds <- function(x){
  x$cut     <- factor(x$cut,
                      levels = levels(diamondLevels$cut), ordered = TRUE)
  x$clarity <- factor(x$clarity,
                      levels = levels(diamondLevels$clarity), ordered = TRUE)
  x$color   <- factor(x$color,
                      levels = levels(diamondLevels$color), ordered = TRUE)
  exp(predict(model, newdata = x))
}


# Publish the service

ws <- workspace()
ep <- publishWebService(ws, fun = predictDiamonds, name = "diamonds",
                        inputSchema = test,
                        data.frame = TRUE
)

# Consume the service
results <- consume(ep, test)$ans
plot(results ~ test$price)

deleteWebService(ws, "diamonds")



# Simple example using scalar input -----------------------------------------

ws <- workspace()
```

```
# Really simple example:
add <- function(x,y) x + y
endpoint <- publishWebService(ws,
                              fun = add,
                              name = "addme",
                              inputSchema = list(x="numeric",
                                                 y="numeric"),
                              outputSchema = list(ans="numeric"))
consume(endpoint, list(x=pi, y=2))

# Now remove the web service named "addme" that we just published
deleteWebService(ws, "addme")



# Send a custom R function for evaluation in AzureML -------------------------

# A neat trick to evaluate any expression in the Azure ML virtual
# machine R session and view its output:
ep <- publishWebService(ws,
                        fun =  function(expr) {
                          paste(capture.output(
                            eval(parse(text=expr))), collapse="\n")
                        },
                        name="commander",
                        inputSchema = list(x = "character"),
                        outputSchema = list(ans = "character"))
cat(consume(ep, list(x = "getwd()"))$ans)
cat(consume(ep, list(x = ".packages(all=TRUE)"))$ans)
cat(consume(ep, list(x = "R.Version()"))$ans)

# Remove the service we just published
deleteWebService(ws, "commander")



# Understanding the scoping rules --------------------------------------------

# The following example illustrates scoping rules. Note that the function
# refers to the variable y defined outside the function body. That value
# will be exported with the service.
y <- pi
ep <- publishWebService(ws,
                        fun = function(x) x + y,
                        name = "lexical scope",
                        inputSchema = list(x = "numeric"),
                        outputSchema = list(ans = "numeric"))
cat(consume(ep, list(x=2))$ans)

# Remove the service we just published
deleteWebService(ws, "lexical scope")
```

```
# Demonstrate scalar inputs but sending a data frame for scoring -------------

# Example showing the use of consume to score all the rows of a data frame
# at once, and other invocations for evaluating multiple sets of input
# values. The columns of the data frame correspond to the input parameters
# of the web service in this example:
f <- function(a,b,c,d) list(sum = a+b+c+d, prod = a*b*c*d)
ep <-  publishWebService(ws,
                         f,
                         name = "rowSums",
                         inputSchema = list(
                           a = "numeric",
                           b = "numeric",
                           c = "numeric",
                           d = "numeric"
                         ),
                         outputSchema = list(
                           sum ="numeric",
                           prod = "numeric")
)
x <- head(iris[,1:4])  # First four columns of iris

# Note the following will FAIL because of a name mismatch in the arguments
# (with an informative error):
consume(ep, x, retryDelay=1)
# We need the columns of the data frame to match the inputSchema:
names(x) <- letters[1:4]
# Now we can evaluate all the rows of the data frame in one call:
consume(ep, x)
# output should look like:
#    sum    prod
# 1 10.2   4.998
# 2  9.5   4.116
# 3  9.4  3.9104
# 4  9.4   4.278
# 5 10.2    5.04
# 6 11.4 14.3208

# You can use consume to evaluate just a single set of input values with this
# form:
consume(ep, a=1, b=2, c=3, d=4)

# or, equivalently,
consume(ep, list(a=1, b=2, c=3, d=4))

# You can evaluate multiple sets of input values with a data frame input:
consume(ep, data.frame(a=1:2, b=3:4, c=5:6, d=7:8))

# or, equivalently, with multiple lists:
consume(ep, list(a=1, b=3, c=5, d=7), list(a=2, b=4, c=6, d=8))

# Remove the service we just published
```

```
deleteWebService(ws, "rowSums")

# A more efficient way to do the same thing using data frame input/output:
f <- function(df) with(df, list(sum = a+b+c+d, prod = a*b*c*d))
ep = publishWebService(ws, f, name="rowSums2",
                       inputSchema = data.frame(a = 0, b = 0, c = 0, d = 0))
consume(ep, data.frame(a=1:2, b=3:4, c=5:6, d=7:8))
deleteWebService(ws, "rowSums2")



# Automatically discover dependencies ----------------------------------------

# The publishWebService function uses `miniCRAN` to include dependencies on
# packages required by your function. The next example uses the `lmer`
# function from the lme4 package, and also shows how to publish a function
# that consumes a data frame by setting data.frame=TRUE.  Note! This example
# depends on a lot of packages and may take some time to upload to Azure.
library(lme4)
# Build a sample mixed effects model on just a subset of the sleepstudy data...
set.seed(1)
m <- lmer(Reaction ~ Days + (Days | Subject),
          data=sleepstudy[sample(nrow(sleepstudy), 120),])
# Deine a prediction function to publish based on the model:
fun <- function(newdata)
{
  predict(m, newdata=newdata)
}
ep <- publishWebService(ws, fun=fun, name="sleepy lmer",
                        inputSchema= sleepstudy,
                        packages="lme4",
                        data.frame=TRUE)

# OK, try this out, and compare with raw data
ans = consume(ep, sleepstudy)$ans
plot(ans, sleepstudy$Reaction)

# Remove the service
deleteWebService(ws, "sleepy lmer")

## End(Not run)
```

`read.AzureML.config` *Reads settings from configuration file in JSON format.*

Description

Reads settings from configuration file in JSON format.

Writes settings to configuration file.

Usage

```
read.AzureML.config(config = getOption("AzureML.config"))

write.AzureML.config(id = NULL, auth = NULL, api_endpoint = NULL,
  management_endpoint = NULL, file = "")
```

Arguments

config	Optional settings file containing id and authorization info. Used if any of the other arguments are missing. The default config file is ~/.azureml/settings.json, but you can change this location by setting options(AzureML.config = "newlocation"). See the section "Using a settings.json file" for more details.
id	Optional workspace id from ML studio -> settings -> WORKSPACE ID. See the section "Finding your AzureML credentials" for more details.
auth	Optional authorization token from ML studio -> settings -> AUTHORIZATION TOKENS. See the section "Finding your AzureML credentials" for more details.
api_endpoint	Optional AzureML API web service URI. Defaults to https://studioap.azureml.net if not provided and not specified in config. See note.
management_endpoint	Optional AzureML management web service URI. Defaults to https://management.azureml.net if not provided and not specified in config. See note.
file	either a character string naming a file or a connection open for writing. "" indicates output to the console.

See Also

write.AzureML.config

workspace

write.AzureML.config

workspace

---

refresh *Refresh data in an AzureML workspace object.*

Description

Contact the AzureML web service and refresh/update data in an AzureML workspace object.

Usage

```
refresh(ws, what = c("everything", "datasets", "experiments", "services"))
```

Arguments

ws	An AzureML workspace reference returned by workspace.
what	Select "everything" to update all cached data, or other values to selectively update those values.

Value

NULL is invisibly returned–this function updates data in the w environment.

See Also

workspace

---

services *List Available Web Services.*

Description

Return a list of web services available to the specified Microsoft Azure Machine Learning workspace. The result is cached in the workspace environment similarly to datasets and experiments.

Usage

```
services(ws, service_id, name, host = ws$.management_endpoint)

getWebServices(ws, service_id, name, host = ws$.management_endpoint)
```

Arguments

ws	An AzureML workspace reference returned by workspace.
service_id	optional web service id. If supplied, return the web service information for just the specified service id. Leave undefined to return a data.frame of all services.
name	optional web service name. If supplied, return the web service information for services with matching names. Leave undefined to return all services.
host	the AzureML web services URI

Value

Returns a data.frame with variables:

- Id
- Name
- Description
- CreationTime
- WorkspaceId
- DefaultEndpointName

Each row of the returned data.frame corresponds to a service.

Note

getWebServices is an alias for services.

See Also

Other discovery functions: discoverSchema; endpointHelp; endpoints, getEndpoints; workspace

Examples

```
## Not run:
workspace_id <- ""          # Your AzureML workspace id
authorization_token <- ""   # Your AzureML authorization token

ws <- workspace(
  id = workspace_id,
  auth = authorization_token
)

# Equivalent:
services(ws)
getWebServices(ws)

## End(Not run)
```

---

upload.dataset *Upload an R data frame to an AzureML workspace.*

Description

Upload any R data frame to an AzureML workspace using the GenericTSV format.

Usage

```
upload.dataset(x, ws, name, description = "", family_id = "", ...)
```

Arguments

x	An R data frame object
ws	An AzureML workspace reference returned by workspace.
name	A character name for the new AzureML dataset (may not match an existing dataset name)
description	An optional character description of the dataset
family_id	An optional AzureML family identifier
...	Optional additional options passed to write.table

Value

A single-row data frame of "Datasets" class that corresponds to the uploaded object now available in ws$datasets.

Note

The additional `write.table` options may not include `sep` or `row.names` or `file`, but any other options are accepted. The AzureML API does not support uploads for _replacing_ datasets with new data by re-using a name. If you need to do this, first delete the dataset from the AzureML Studio interface, then upload a new version.

See Also

Other dataset functions: `datasets`; `delete.datasets`; `download.intermediate.dataset`; `workspace`

Examples

```
## Not run:
  library(AzureML)

  ws <- workspace()

  # Upload the R airquality data.frame to the workspace.
  upload.dataset(airquality, ws, "airquality")

  # Example datasets (airquality should be among them now)
  head(datasets(ws))

  # Now delete what we've just uploaded
  delete.datasets(ws, "airquality")

## End(Not run)
```

`workspace` *Create a reference to an AzureML Studio workspace.*

Description

Create a reference to an AzureML Studio workspace, returning a `Workspace` object that is an R environment containing details and data associated with the AzureML work space. Data corresponding to services, experiments, and datasets in the workspace are cached in the result object environment. See `refresh` about updating cached data.

Usage

```
workspace(id, auth, api_endpoint, management_endpoint,
  config = getOption("AzureML.config"), ..., .validate = TRUE)
```

Arguments

`id`	Optional workspace id from ML studio -> settings -> WORKSPACE ID. See the section "Finding your AzureML credentials" for more details.
`auth`	Optional authorization token from ML studio -> settings -> AUTHORIZATION TOKENS. See the section "Finding your AzureML credentials" for more details.
`api_endpoint`	Optional AzureML API web service URI. Defaults to `https://studioap.azureml.net` if not provided and not specified in config. See note.
`management_endpoint`	Optional AzureML management web service URI. Defaults to `https://management.azureml.net` if not provided and not specified in config. See note.
`config`	Optional settings file containing id and authorization info. Used if any of the other arguments are missing. The default config file is `~/.azureml/settings.json`, but you can change this location by setting `options(AzureML.config = "newlocation")`. See the section "Using a settings.json file" for more details.
`...`	ignored
`.validate`	If TRUE, makes a request to the AzureML API to retrieve some data. This validates whether the workspace id and authorization token are valid. Specifically, the function calls `datasets`. This should normally be set to TRUE. Set this to FALSE for testing, or if you know that your credentials are correct and you don't want to retrieve the datasets.

Value

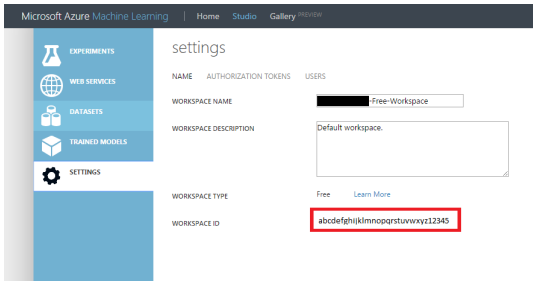
An R environment of class `Workspace` containing at least the following objects:

- experiments: Collection of experiments in the workspace represented as an `Experiments` object. See `experiments`
- datasets: Collection of datasets in the workspace represented as a `Datasets` object. See `datasets`
- services: Collection of web services in the workspace represented as a `Services` object. See `services`

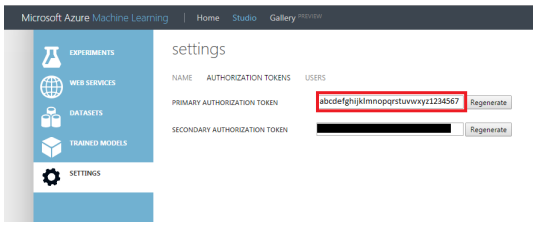
Finding your AzureML credentials

You can find your Azure Machine Learning `workspace id` and `authorization token` in the Azure Machine Learning Studio interface.

Workspace ID



Authorization token



Using a `settings.json` file

If any of the `id`, `auth`, `api_endpoint` or `management_endpoint` arguments are missing, the function attempts to read values from the `config` file with JSON format:

```
{"workspace":{
  "id": "enter your AzureML workspace id here",
  "authorization_token": "enter your AzureML authorization token here",
  "api_endpoint": "https://studioapi.azureml.net",
  "management_endpoint": "https://management.azureml.net"
}}
```

See Also

`datasets`, `experiments`, `refresh`, `services`, `consume`, `publishWebService`

Other consumption functions: `consume`

Other dataset functions: `datasets`; `delete.datasets`; `download.intermediate.dataset`; `upload.dataset`

Other discovery functions: `discoverSchema`; `endpointHelp`; `endpoints`, `getEndpoints`; `getWebServices`, `services`

Other experiment functions: `download.intermediate.dataset`; `experiments`

Other publishing functions: `deleteWebService`; `publishWebService`, `updateWebService`

Index