Package 'AnVIL'

April 10, 2023

Title Bioconductor on the AnVIL compute environment

Version 1.10.2

Description The AnVIL is a cloud computing resource developed in part by the National Human Genome Research Institute. The AnVIL package provides end-user and developer functionality. For the end-user, AnVIL provides fast binary package installation, utitlities for working with Terra / AnVIL table and data resources, and convenient functions for file movement to and from Google cloud storage. For developers, AnVIL provides programatic access to the Terra, Leonardo, Rawls, Dockstore, and Gen3 RESTful programming interface, including helper functions to transform JSON responses to formats more amenable to manipulation in R.

License Artistic-2.0

Encoding UTF-8

Depends R (>= 3.6), dplyr

Imports stats, utils, methods, futile.logger, jsonlite, httr, rapiclient (>= 0.1.3), tibble, tidyselect, tidyr, rlang, shiny, DT, miniUI, htmltools, BiocManager

Suggests parallel, knitr, rmarkdown, testthat, withr, readr, BiocStyle, devtools

Collate utilities.R install.R authenticate.R api.R Service.R Services.R Leonardo.R Terra.R Rawls.R Dockstore.R Gen3.R Response.R gcloud_sdk.R gcloud.R gsutil.R localize.R drs.R av.R avworkspace.R avworkflow.R avworkflow_configuration.R avnotebooks.R gadgets.R zzz.R

VignetteBuilder knitr

biocViews Infrastructure

RoxygenNote 7.2.3

Roxygen list(markdown = TRUE)

git_url https://git.bioconductor.org/packages/AnVIL

git_branch RELEASE_3_16

git_last_commit 38838f0

git_last_commit_date 2023-03-23

Date/Publication 2023-04-10

Author Martin Morgan [aut, cre] (<https://orcid.org/0000-0002-5874-8148>),
Kayla Interdonato [aut],
Nitesh Turaga [aut],
BJ Stubbs [ctb],
Vincent Carey [ctb],
Marcel Ramos [ctb],
Sehyun Oh [ctb],
Sweta Gopaulakrishnan [ctb],
Valerie Obenchain [ctb]

Maintainer Martin Morgan <mtmorgan.bioc@gmail.com>

R topics documented:

`.gadget_run` *Functions to implement AnVIL gadget interfaces*

Description

Functions documented on this page are primarily intended for package developers wishing to implement gadgets (graphical interfaces) to navigating AnVIL-generated tables.

`.gadget_run()` presents the user with a tibble-navigating gadget, returning the value of `DONE_FUN` if a row of the tibble is selected, or NULL.

Usage

```
.gadget_run(title, tibble, DONE_FUN)
```

Arguments

`title`	character(1) (required) title to appear at the base of the gadget, e.g., "AnVIL Workspaces".
`tibble`	a `tibble` or `data.frame` to be displayed in the gadget.
`DONE_FUN`	a function of two arguments, `tibble` and `row_selected`. The tibble is the `tibble` provided as an argument to `.gadget_run()`. `row_selected` is the row selected in the gadget by the user. The function is only invoked when the user selects a valid row.

Value

`.gadget_run()` returns the result of `DONE_FUN()` if a row has been selected by the user, or `NULL` if no row is selected (the user presses `Cancel`, or `Done` prior to selecting any row).

Examples

```
## Not run:
tibble <- avworkspaces()
DONE_FUN <- function(tibble, row_selected) {
    selected <- slice(tibble, row_selected)
    with(selected, paste0(namespace, "/", name))
}
.gadget_run("AnVIL Example", tibble, DONE_FUN)

## End(Not run)
```

`AnVIL-deprecated` *Deprecated functions in package ‘AnVIL’*

Description

These functions are provided for compatibility with older versions of ‘AnVIL’ only, and will be defunct at the next release.

Usage

```
avworkflow_configuration(
  configuration_namespace,
  configuration_name,
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)
```

```
avworkflow_import_configuration(
  config,
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)
```

Arguments

`configuration_namespace`	character(1).
`configuration_name`	character(1).
`namespace`	character(1).
`name`	character(1).
`config`	`avworkflow_configuration` object.

Details

The following functions are deprecated and will be made defunct; use the replacement indicated below:

- `avworkflow_configuration()`: `avworkflow_configuration_get`
- `avworkflow_import_configuration()`: `avworkflow_configuration_set`

`av` *TABLE, DATA, files, bucket, runtime, and disk elements*

Description

`avtables()` describes tables available in a workspace. Tables can be visualized under the DATA tab, TABLES item. `avtable()` returns an AnVIL table. `avtable_paged()` retrieves an AnVIL table by requesting the table in 'chunks', and may be appropriate for large tables. `avtable_import()` imports a data.frame to an AnVIL table. `avtable_import_set()` imports set membership (i.e., a subset of an existing table) information to an AnVIL table. `avtable_delete_values()` removes rows from an AnVIL table.

`avdata()` returns key-value tables representing the information visualized under the DATA tab, 'REFERENCE DATA' and 'OTHER DATA' items. `avdata_import()` updates (modifies or creates new, but does not delete) rows in 'REFERENCE DATA' or 'OTHER DATA' tables.

`avbucket()` returns the workspace bucket, i.e., the google bucket associated with a workspace. Bucket content can be visualized under the 'DATA' tab, 'Files' item.

`avfiles_ls()` returns the paths of files in the workspace bucket. `avfiles_backup()` copies files from the compute node file system to the workspace bucket. `avfiles_restore()` copies files from the workspace bucket to the compute node file system. `avfiles_rm()` removes files or directories from the workspace bucket.

`avruntimes()` returns a tibble containing information about runtimes (notebooks or RStudio instances, for example) that the current user has access to.

`avruntime()` returns a tibble with the runtimes associated with a particular google project and account number; usually there is a single runtime satisfiying these criteria, and it is the runtime active in AnVIL.

'avdisks()‘ returns a tibble containing information about persistent disks associatd with the current user.

Usage

```
avtables(namespace = avworkspace_namespace(), name = avworkspace_name())

avtable(table, namespace = avworkspace_namespace(), name = avworkspace_name())

avtable_paged(
  table,
  n = Inf,
  page = 1L,
  pageSize = 1000L,
  sortField = "name",
  sortDirection = c("asc", "desc"),
  filterTerms = character(),
  filterOperator = c("and", "or"),
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)

avtable_import(
  .data,
  entity = names(.data)[[1]],
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  delete_empty_values = FALSE
)

avtable_import_set(
  .data,
  origin,
  set = names(.data)[[1]],
  member = names(.data)[[2]],
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  delete_empty_values = FALSE
)

avtable_delete_values(
  table,
  values,
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)

avdata(namespace = avworkspace_namespace(), name = avworkspace_name())

avdata_import(
  .data,
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)

avbucket(
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  as_path = TRUE
)

avfiles_ls(
  path = "",
  full_names = FALSE,
  recursive = FALSE,
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)

avfiles_backup(
  source,
  destination = "",
  recursive = FALSE,
  parallel = TRUE,
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)

avfiles_restore(
  source,
  destination = ".",
  recursive = FALSE,
  parallel = TRUE,
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)

avfiles_rm(
  source,
  recursive = FALSE,
  parallel = TRUE,
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)

avruntimes()

avruntime(project = gcloud_project(), account = gcloud_account())

avdisks()
```

Arguments

`namespace`	character(1) AnVIL workspace namespace as returned by, e.g., `avworkspace_namespace()`
`name`	character(1) AnVIL workspace name as returned by, eg., `avworkspace_name()`.
`table`	character(1) table name as returned by, e.g., `avtables()`.
`n`	numeric(1) maximum number of rows to return
`page`	integer(1) first page of iteration
`pageSize`	integer(1) number of records per page. Generally, larger page sizes are more efficient.
`sortField`	character(1) field used to sort records when determining page order. Default is the entity field.
`sortDirection`	character(1) direction to sort entities (`"asc"`ending or `"desc"`ending) when paging.
`filterTerms`	character(1) string literal to select rows with an exact (substring) matches in column.
`filterOperator`	character(1) operator to use when multiple terms in `filterTerms=`, either `"and"` (default) or `"or"`.
`.data`	A tibble or data.frame for import as an AnVIL table.
`entity`	`character(1)` column name of `.data` to be used as imported table name. When the table comes from R, this is usually a column name such as `sample`. The data will be imported into AnVIL as a table `sample`, with the `sample` column included with suffix `_id`, e.g., `sample_id`. A column in `.data` with suffix `_id` can also be used, e.g., `entity = "sample_id"`, creating the table `sample` with column `sample_id` in AnVIL. Finally, a value of `entity` that is not a column in `.data`, e.g., `entity = "unknown"`, will cause a new table with name `entity` and entity values `seq_len(nrow(.data))`.
`delete_empty_values`	logical(1) when `TRUE`, remove entities not include in `.data` from the DATA table. Default: `FALSE`.
`origin`	character(1) name of the entity (table) used to create the set e.g "sample", "participant", etc.
`set`	`character(1)` column name of `.data` identifying the set(s) to be created.
`member`	`character()` vector of entity from the avtable identified by `origin`. The values may repeat if an ID is in more than one set

`values`	vector of values in the entity (key) column of `table` to be deleted. A table `sample` has an associated entity column with suffix `_id`, e.g., `sample_id`. Rows with entity column entries matching `values` are deleted.
`as_path`	logical(1) when TRUE (default) return bucket with prefix `gs://` (for `avbucket()`) or `gs://<bucket-id>` (for `avfiles_ls()`).
`path`	For `avfiles_ls(), the character(1) file or directory path to list. For av-`files_rm()`, the character() (perhaps with length greater than 1) of files or directory pat`
`full_names`	logical(1) return names relative to `path` (`FALSE`, default) or root of the workspace bucket?
`recursive`	logical(1) list files recursively?
`source`	character() file paths. for `avfiles_backup()`, `source` can include directory names when `recursive = TRUE`.
`destination`	character(1) a google bucket (`gs://<bucket-id>/...`) to write files. The default is the workspace bucket.
`parallel`	logical(1) backup files using parallel transfer? See `?gsutil_cp()`.
`project`	`character(1)` project (billing account) name, as returned by, e.g., `gcloud_project()` or `avworkspace_namespace()`.
`account`	`character(1)` google account (email address associated with billing account), as returned by `gcloud_account()`.

Details

`avtable_import_set()` creates new rows in a table `<origin>_set`. One row will be created for each distinct value in the column identified by `set`. Each row entry has a corresponding column `<origin>` linking to one or more rows in the `<origin>` table, as given in the `member` column. The operation is somewhat like `split(member, set)`.

`avfiles_backup()` can be used to back-up individual files or entire directories, recursively. When `recursive = FALSE`, files are backed up to the bucket with names approximately `paste0(destination, "/", basename(source))`. When `recursive = TRUE` and source is a directory `path/to/foo/', files are backed up to b` "/", dir(basename(source), full.names = TRUE))`. Naming conventions are described in detail in gsu-`til_help("cp")‘.

`avfiles_restore()` behaves in a manner analogous to `avfiles_backup()`, copying files from the workspace bucket to the compute node file system.

Value

`avtables()`: A tibble with columns identifying the table, the number of records, and the column names.

`avtable()`: a tibble of data corresponding to the AnVIL table `table` in the specified workspace.

`avtable_paged()`: a tibble of data corresponding to the AnVIL table `table` in the specified workspace.

`avtable_import()` returns a `character(1)` name of the imported AnVIL tibble.

`avtable_import_set()` returns a `character(1)` name of the imported AnVIL tibble.

`avtable_delete_values()` returns a `tibble` representing deleted entities, invisibly.

`avdata()` returns a tibble with five columns: `"type"` represents the origin of the data from the 'REFERENCE' or 'OTHER' data menus. `"table"` is the table name in the `REFERENCE` menu, or 'workspace' for the table in the 'OTHER' menu, the key used to access the data element, the value label associated with the data element and the value (e.g., google bucket) of the element.

`avdata_import()` returns, invisibly, the subset of the input table used to update the AnVIL tables.

`avbucket()` returns a `character(1)` bucket identifier, prefixed with `gs://` if `as_path = TRUE`.

`avfiles_ls()` returns a character vector of files in the workspace bucket.

`avfiles_backup()` returns, invisibly, the status code of the `gsutil_cp()` command used to back up the files.

`avfiles_rm()` on success, returns a list of the return codes of `gsutil_rm()`, invisibly.

`avruntimes()` returns a tibble with columns

- id: integer() runtime identifier.
- googleProject: character() billing account.
- tool: character() e.g., "Jupyter", "RStudio".
- status character() e.g., "Stopped", "Running".
- creator character() AnVIL account, typically "user@gmail.com".
- createdDate character() creation date.
- destroyedDate character() destruction date, or NA.
- dateAccessed character() date of (first?) access.
- runtimeName character().
- clusterServiceAccount character() service ('pet') account for this runtime.
- masterMachineType character() It is unclear which 'tool' populates which of the machineType columns).
- workerMachineType character().
- machineType character().
- persistentDiskId integer() identifier of persistent disk (see `avdisks()`), or `NA`.

`avruntime()` returns a tibble witht he same structure as the return value of `avruntimes()`.

`avdisks()` returns a tibble with columns

- id character() disk identifier.
- googleProject: character() billing account.
- status, e.g, "Ready"
- size integer() in GB.
- diskType character().
- blockSize integer().
- creator character() AnVIL account, typically "user@gmail.com".
- createdDate character() creation date.
- destroyedDate character() destruction date, or NA.
- dateAccessed character() date of (first?) access.
- zone character() e.g.. "us-central1-a".
- name character().

Examples

```
## Not run:
## editable copy of '1000G-high-coverage-2019' workspace
avworkspace("bioconductor-rpci-anvil/1000G-high-coverage-2019")
sample <-
    avtable("sample") %>%                               # existing table
    mutate(set = sample(head(LETTERS), nrow(.), TRUE))  # arbitrary groups
sample %>%                                  # new 'participant_set' table
    avtable_import_set("participant", "set", "participant")
sample %>%                                  # new 'sample_set' table
    avtable_import_set("sample", "set", "name")

## End(Not run)

if (gcloud_exists() && nzchar(avworkspace_name())) {
    ## from within AnVIL
    data <- avdata()
    data
}

## Not run:
avdata_import(data)

## End(Not run)

if (gcloud_exists() && nzchar(avworkspace_name()))
    ## From within AnVIL...
    bucket <- avbucket()                        # discover bucket

## Not run:
path <- file.path(bucket, "mtcars.tab")
gsutil_ls(dirname(path))                    # no 'mtcars.tab'...
write.table(mtcars, gsutil_pipe(path, "w")) # write to bucket
gsutil_stat(path)                           # yep, there!
read.table(gsutil_pipe(path, "r"))          # read from bucket

## End(Not run)
if (gcloud_exists() && nzchar(avworkspace_name()))
    avfiles_ls()

## Not run:
## backup all files in the current directory
## default buckets are gs://<bucket-id>/<file-names>
avfiles_backup(dir())
## backup working directory, recursively
## default buckets are gs://<bucket-id>/<basename(getwd())>/...
avfiles_backup(getwd(), recursive = TRUE)

## End(Not run)

if (gcloud_exists())
    ## from within AnVIL
    avruntimes()

if (gcloud_exists())
    ## from within AnVIL
    avdisks()
```

`avnotebooks` *Notebook management*

Description

`avnotebooks()` returns the names of the notebooks associated with the current workspace.

`avnotebooks_localize()` synchronizes the content of the workspace bucket to the local file system.

`avnotebooks_delocalize()` synchronizes the content of the notebook location of the local file system to the workspace bucket.

Usage

```
avnotebooks(
  local = FALSE,
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)

avnotebooks_localize(
  destination,
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  dry = TRUE
)

avnotebooks_delocalize(
  source,
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  dry = TRUE
)
```

Arguments

`local`	= `logical(1)` notebooks located on the workspace (`local = FALSE`, default) or runtime / local instance (`local = TRUE`). When `local = TRUE`, the notebook path is `<avworkspace_name>/notebooks`.
`namespace`	character(1) AnVIL workspace namespace as returned by, e.g., `avworkspace_namespace()`

`name`	character(1) AnVIL workspace name as returned by, eg., `avworkspace_name()`.
`destination`	missing or character(1) file path to the local file system directory for synchronization. The default location is `~/<avworkspace_name>/notebooks`. Out-of-date local files are replaced with the workspace version.
`dry`	`logical(1)`, when `TRUE` (default), return the consequences of the operation without actually performing the operation.
`source`	missing or character(1) file path to the local file system directory for synchronization. The default location is `~/<avworkspace_name>/notebooks`. Out-of-date local files are replaced with the workspace version.

Value

`avnotebooks()` returns a character vector of buckets / files located in the workspace 'Files/notebooks' bucket path, or on the local file system.

`avnotebooks_localize()` returns the exit status of `gsutil_rsync()`.

`avnotebooks_delocalize()` returns the exit status of `gsutil_rsync()`.

Examples

```
if (gcloud_exists() && nzchar(avworkspace_name()))
    avnotebooks()

if (gcloud_exists() && nzchar(avworkspace_name()))
    avnotebooks_localize()  # dry run

if (gcloud_exists() && nzchar(avworkspace_name()))
    try(avnotebooks_delocalize())  # dry run, fails if no local resource
```

`avworkflows` *Workflow submissions and file outputs*

Description

`avworkflows()` returns a tibble summarizing available workflows.

`avworkflow_jobs()` returns a tibble summarizing submitted workflow jobs for a namespace and name.

`avworkflow_files()` returns a tibble containing information and file paths to workflow outputs.

`avworkflow_localize()` creates or synchronizes a local copy of files with files stored in the workspace bucket and produced by the workflow.

`avworkflow_run()` runs the workflow of the configuration.

`avworkflow_stop()` stops the most recently submitted workflow jub from running.

Usage

```
avworkflows(namespace = avworkspace_namespace(), name = avworkspace_name())

avworkflow_jobs(namespace = avworkspace_namespace(), name = avworkspace_name())

avworkflow_files(
  submissionId = NULL,
  bucket = avbucket(),
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)

avworkflow_localize(
  submissionId = NULL,
  destination = NULL,
  type = c("control", "output", "all"),
  bucket = avbucket(),
  dry = TRUE
)

avworkflow_run(
  config,
  entityName,
  entityType = config$rootEntityType,
  deleteIntermediateOutputFiles = FALSE,
  useCallCache = TRUE,
  useReferenceDisks = FALSE,
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  dry = TRUE
)

avworkflow_stop(
  submissionId = NULL,
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  dry = TRUE
)
```

Arguments

`namespace`	character(1) AnVIL workspace namespace as returned by, e.g., `avworkspace_namespace()`
`name`	character(1) AnVIL workspace name as returned by, eg., `avworkspace_name()`.
`submissionId`	a character() of workflow submission ids, or a tibble with column `submissionId`, or NULL / missing. See 'Details'.
`bucket`	character(1) DEPRECATED (ignored in the current release) name of the google bucket in which the workflow products are available, as `gs://...`. Usually the bucket of the active workspace, returned by `avbucket()`.

`destination`	character(1) file path to the location where files will be synchronized. For directories in the current working directory, be sure to prepend with `"./"`. When `NULL`, the `submissionId` is used as the destination. `destination` may also be a google bucket, in which case th workflow files are synchronized from the workspace to a second bucket.
`type`	character(1) copy `"control"` (default), `"output"`, or `"all"` files produced by a workflow.
`dry`	logical(1) when `TRUE` (default), report the consequences but do not perform the action requested. When `FALSE`, perform the action.
`config`	a `avworkflow_configuration` object of the workflow that will be run.
`entityName`	character(1) or NULL name of the set of samples to be used when running the workflow. NULL indicates that no sample set will be used.
`entityType`	character(1) or NULL type of root entity used for the workflow. NULL means that no root entity will be used.
`deleteIntermediateOutputFiles`	logical(1) whether or not to delete intermediate output files when the workflow completes.
`useCallCache`	logical(1) whether or not to read from cache for this submission.
`useReferenceDisks`	logical(1) whether or not to use pre-built disks for common genome references. Default: `FALSE`.

Details

For `avworkflow_files()`, the `submissionId` is the identifier associated with the submission of one (or more) workflows, and is present in the return value of `avworkflow_jobs()`; the example illustrates how the first row of `avworkflow_jobs()` (i.e., the most recently completed workflow) can be used as input to `avworkflow_files()`. When `submissionId` is not provided, the return value is for the most recently submitted workflow of the namespace and name of `avworkspace()`.

`avworkflow_localize()`. `type = "control"` files summarize workflow progress; they can be numerous but are frequently small and quickly syncronized. `type = "output"` files are the output products of the workflow stored in the workspace bucket. Depending on the workflow, outputs may be large, e.g., aligned reads in bam files. See `gsutil_cp()` to copy individual files from the bucket to the local drive.

`avworkflow_localize()` treats `submissionId=` in the same way as `avworkflow_files()`: when missing, files from the most recent workflow job are candidates for localization.

Value

`avworkflows()` returns a tibble. Each workflow is in a 'namespace' and has a 'name', as illustrated in the example. Columns are

- name: workflow name.
- namespace: workflow namespace (often the same as the workspace namespace).
- rootEntityType: name of the `avtable()` used to retrieve inputs.
- methodRepoMethod.methodUri: source of the method, e.g., a dockstore URI.

- methodRepoMethod.sourceRepo: source repository, e.g., dockstore.
- methodRepoMethod.methodPath: path to method, e.g., a dockerstore method might reference a github repository.
- methodRepoMethod.methodVersion: the version of the method, e.g., 'main' branch of a github repository.

`avworkflow_jobs()` returns a tibble, sorted by `submissionDate`, with columns

- submissionId character() job identifier from the workflow runner.
- submitter character() AnVIL user id of individual submitting the job.
- submissionDate POSIXct() date (in local time zone) of job submission.
- status character() job status, with values 'Accepted' 'Evaluating' 'Submitting' 'Submitted' 'Aborting' 'Aborted' 'Done'
- succeeded integer() number of workflows succeeding.
- failed integer() number of workflows failing.

`avworkflow_files()` returns a tibble with columns

- file: character() 'base name' of the file in the bucket.
- workflow: character() name of the workflow the file is associated with.
- task: character() name of the task in the workflow that generated the file.
- path: charcter() full path to the file in the google bucket.
- submissionId: character() internal identifier associated with the submission the files belong to.
- workflowId: character() internal identifer associated with each workflow (e.g., row of an avtable() used as input) in the submission.
- submissionRoot: character() path in the workspace bucket to the root of files created by this submission.
- namespace: character() AnVIL workspace namespace (billing account) associated with the submissionId.
- name: character(1) AnVIL workspace name associated with the submissionId.

`avworkflow_localize()` prints a message indicating the number of files that are (if `dry = FALSE`) or would be localized. If no files require localization (i.e., local files are not older than the bucket files), then no files are localized. `avworkflow_localize()` returns a tibble of file name and bucket path of files to be synchronized.

`avworkflow_run()` returns `config`, invisibly.

`avworkflow_stop()` returns (invisibly) `TRUE` on successfully requesting that the workflow stop, `FALSE` if the workflow is already aborting, aborted, or done.

Examples

```
if (gcloud_exists() && nzchar(avworkspace_name()))
    ## from within AnVIL
    avworkflows() %>% select(namespace, name)

if (gcloud_exists() && nzchar(avworkspace_name()))
```

```
    ## from within AnVIL
    avworkflow_jobs()

if (gcloud_exists() && nzchar(avworkspace_name())) {
    ## e.g., from within AnVIL
    avworkflow_jobs() |>
    ## select most recent workflow
    head(1) |>
    ## find paths to output and log files on the bucket
    avworkflow_files()
}

if (gcloud_exists() && nzchar(avworkspace_name())) {
    avworkflow_localize(dry = TRUE)
}

## Not run:
entityName <- avtable("participant_set") |>
    pull(participant_set_id) |>
    head(1)
avworkflow_run(new_config, entityName)

## End(Not run)

## Not run:
avworkflow_stop()

## End(Not run)
```

---

avworkflow_configurations

Workflow configuration

Description

Funtions on this help page facilitate getting, updating, and settting workflow configuration parameters. See ?avworkflow for additional relevant functionality.

avworkflow_namespace() and avworkflow_name() are utility functions to record the workflow namespace and name required when working with workflow configurations. avworkflow() provides a convenient way to provide workflow namespace and name in a single command, namespace/name.

avworkflow_configuration_get() returns a list structure describing an existing workflow configuration.

avworkflow_configuration_inputs() returns a data.frame template for the inputs defined in a workflow configuration. This template can be used to provide custom inputs for a configuration.

avworkflow_configuration_outputs() returns a data.frame template for the outputs defined in a workflow configuration. This template can be used to provide custom outputs for a configuration.

avworkflow_configuration_update() returns a list structure describing a workflow configuration with updated inputs and / or outputs.

avworkflow_configuration_set() updates an existing configuration, e.g., changing inputs to the workflow.

avworkflow_configuration_template() returns a template for defining workflow configurations. This template can be used as a starting point for providing a custom configuration.

Usage

```
avworkflow_namespace(workflow_namespace = NULL)

avworkflow_name(workflow_name = NULL)

avworkflow(workflow = NULL)

avworkflow_configuration_get(
  workflow_namespace = avworkflow_namespace(),
  workflow_name = avworkflow_name(),
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)

avworkflow_configuration_inputs(config)

avworkflow_configuration_outputs(config)

avworkflow_configuration_update(
  config,
  inputs = avworkflow_configuration_inputs(config),
  outputs = avworkflow_configuration_outputs(config)
)

avworkflow_configuration_set(
  config,
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  dry = TRUE
)

avworkflow_configuration_template()

## S3 method for class 'avworkflow_configuration'
print(x, ...)
```

Arguments

workflow_namespace	character(1) AnVIL workflow namespace, as returned by, e.g., the namespace column of avworkflows().

`workflow_name`	character(1) AnVIL workflow name, as returned by, e.g., the `name` column of `avworkflows()`.
`workflow`	character(1) representing the combined workflow namespace and name, as `namespace/name`.
`namespace`	character(1) AnVIL workspace namespace as returned by, e.g., `avworkspace_namespace()`
`name`	character(1) AnVIL workspace name as returned by, eg., `avworkspace_name()`.
`config`	a named list describing the full configuration, e.g., created from editing the return value of `avworkflow_configuration_set()` or `avworkflow_configuration_template()`.
`inputs`	the new inputs to be updated in the workflow configuration. If none are specified, the inputs from the original configuration will be used and no changes will be made.
`outputs`	the new outputs to be updated in the workflow configuration. If none are specified, the outputs from the original configuration will be used and no changes will be made.
`dry`	logical(1) when `TRUE` (default), report the consequences but do not perform the action requested. When `FALSE`, perform the action.
`x`	Object of class `avworkflow_configuration`.
`...`	additional arguments to `print()`; unused.

Details

The exact format of the configuration is important.

One common problem is that a scalar character vector `"bar"` is interpretted as a json 'array' `["bar"]` rather than a json string `"bar"`. Enclose the string with `jsonlite::unbox("bar")` in the configuration list if the length 1 character vector in R is to be interpretted as a json string.

A second problem is that an unquoted unboxed character string `unbox("foo")` is required by AnVIL to be quoted. This is reported as a warning() about invalid inputs or outputs, and the solution is to provide a quoted string `unbox('"foo"')`.

Value

`avworkflow_namespace()`, and `avworkflow_name()` return `character(1)` identifiers. `avworkflow()` returns the character(1) concatenated namespace and name. The value returned by `avworkflow_name()` will be percent-encoded (e.g., spaces `" "` replaced by `"%20"`).

`avworkflow_configuration_get()` returns a list structure describing the configuration. See `avworkflow_configuration_` for the structure of a typical workflow.

`avworkflow_configuration_inputs()` returns a data.frame providing a template for the configuration inputs, with the following columns:

- inputType
- name
- optional
- attribute

The only column of interest to the user is the attribute column, this is the column that should be changed for customization.

`avworkflow_configuration_outputs()` returns a data.frame providing a template for the configuration outputs, with the following columns: - name - outputType - attribute

The only column of interest to the user is the attribute column, this is the column that should be changed for customization.

`avworkflow_configuration_update()` returns a list structure describing the updated configuration.

`avworkflow_configuration_set()` returns an object describing the updated configuration. The return value includes invalid or unused elements of the `config` input. Invalid or unused elements of `config` are also reported as a warning.

`avworkflow_configuration_template()` returns a list providing a template for configuration lists, with the following structure:

- namespace character(1) configuration namespace.
- name character(1) configuration name.
- rootEntityType character(1) or missing. the name of the table (from `avtables()`) containing the entitites referenced in inputs, etc., by the keyword 'this.'
- prerequisites named list (possibly empty) of prerequisites.
- inputs named list (possibly empty) of inputs. Form of input depends on method, and might include, e.g., a reference to a field in a table referenced by `avtables()` or a character string defining an input constant.
- outputs named list (possibly empty) of outputs.
- methodConfigVersion integer(1) identifier for the method configuration.
- methodRepoMethod named list describing the method, with character(1) elements described in the return value for `avworkflows()`.
  - methodUri
  - sourceRepo
  - methodPath
  - methodVersion. The REST specification indicates that this has type `integer`, but the documentation indicates either `integer` or `string`.
- deleted logical(1) of uncertain purpose.

See Also

The help page ?`avworkflow` for discovering, running, stopping, and retrieving outputs from workflows.

Examples

```
## set the namespace and name as appropriate
avworkspace("bioconductor-rpci-anvil/Bioconductor-Workflow-DESeq2")

## discover available workflows in the workspace
if (gcloud_exists())
```

```
    avworkflows()

## record the workflow of interest
avworkflow("bioconductor-rpci-anvil/AnVILBulkRNASeq")

## what workflows are available?
if (gcloud_exists()) {
    available_workflows <- avworkflows()

    ## retrieve the current configuration
    config <- avworkflow_configuration_get()
    config

    ## what are the inputs and outputs?
    inputs <- avworkflow_configuration_inputs(config)
    inputs

    outputs <- avworkflow_configuration_outputs(config)
    outputs

    ## update inputs or outputs, e.g., this input can be anything...
    inputs <-
        inputs |>
        mutate(attribute = ifelse(
            name == "salmon.transcriptome_index_name",
            '"new_index_name"',
            attribute
        ))
    new_config <- avworkflow_configuration_update(config, inputs)
    new_config

    ## set the new configuration in AnVIL; use dry = FALSE to actually
    ## update the configuration
    avworkflow_configuration_set(config)
}

## avworkflow_configuration_template() is a utility function that may
## help understanding what the inputs and outputs should be
avworkflow_configuration_template() |>
    str()

avworkflow_configuration_template()
```

---

avworkspace *Workspace management*

Description

avworkspaces() returns a tibble with available workspaces.

`avworkspace_namespace()` and `avworkspace_name()` are utiliity functions to retrieve workspace namespace and name from environment variables or interfaces usually available in AnVIL notebooks or RStudio sessions. `avworkspace()` provides a convenient way to specify workspace namespace and name in a single command.

`avworkspace_clone()` clones (copies) an existing workspace, possibly into a new namespace (billing account).

Usage

```
avworkspaces()

avworkspace_namespace(namespace = NULL, warn = TRUE)

avworkspace_name(name = NULL, warn = TRUE)

avworkspace(workspace = NULL)

avworkspace_clone(
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  to_namespace = namespace,
  to_name,
  bucket_location = "US"
)
```

Arguments

`namespace`	character(1) AnVIL workspace namespace as returned by, e.g., `avworkspace_namespace()`
`warn`	logical(1) when `TRUE` (default), generate a warning when the workspace namespace or name cannot be determined.
`name`	character(1) AnVIL workspace name as returned by, eg., `avworkspace_name()`.
`workspace`	when present, a `character(1)` providing the concatenated namespace and name, e.g., `"bioconductor-rpci-anvil/Bioconductor-Package-AnVIL"`
`to_namespace`	character(1) workspace (billing account) in which to make the clone.
`to_name`	character(1) name of the cloned workspace.
`bucket_location`	character(1) region (NO multi-region, except the default) in which bucket attached to the workspace should be created.

Details

`avworkspace_namespace()` is the billing account. If the `namespace=` argument is not provided, try `gcloud_project()`, and if that fails try `Sys.getenv("WORKSPACE_NAMESPACE")`.

`avworkspace_name()` is the name of the workspace as it appears in `https://app.terra.bio/#workspaces`. If not provided, `avworkspace_name()` tries to use `Sys.getenv("WORKSPACE_NAME")`.

Namespace and name values are cached across sessions, so explicitly providing `avworkspace_name*()` is required at most once per session. Revert to system settings with arguments `NA`.

Value

`avworkspaces()` returns a tibble with columns including the name, last modification time, namespace, and owner status.

`avworkspace_namespace()`, and `avworkspace_name()` return `character(1)` identifiers. `avworkspace()` returns the character(1) concatenated namespace and name. The value returned by `avworkspace_name()` will be percent-encoded (e.g., spaces `" "` replaced by `"%20"`).

`avworkspace_clone()` returns the namespace and name, in the format `namespace/name`, of the cloned workspace.

Examples

```
avworkspace_namespace()
avworkspace_name()
avworkspace()
```

`avworkspace_gadget` *Graphical user interfaces for common AnVIL operations*

Description

`workspace()` allows choice of workspace for subsequent use. It is the equivalent of displaying workspaces with `avworkspaces()`, and setting the selected workspace with `avworkspace()`.

`browse_workspace()` uses `browseURL()` to open a browser window pointing to the Terra workspace.

`table()` allows choice of table in the current workspace (selected by `avworkspace()` or `workspace()`) to be returned as a tibble. It is equivalent to invoking `avtables()` to show available tables, and `avtable()` to retrieve the selected table.

`workflow()` allows choice of workflow for retrieval. It is the equivalent of `avworkflows()` for listing available workflows, and `avworkflow_configuration_get()` for retrieving the workflow.

Usage

```
avworkspace_gadget()

browse_workspace(use_avworkspace = TRUE)

avtable_gadget()

avworkflow_gadget()
```

Arguments

`use_avworkspace`	logical(1) when `TRUE` (default), use the selected workspace (via `workspace()` or `avworkspace()` if available. If `FALSE` or no workspace is currently selected, use `workspace()` to allow the user to select the workspace.

Value

`workspace()` returns the selected workspace as a character(1) using the format namespace/name, or character(0) if no workspace is selected.

`browse_workspace()` returns the status of a `system()` call to launch the browser, invisibly.

`table()` returns a `tibble` representing the selected AnVIL table.

`workflow()` returns an `avworkflow_configuration` object representing the inputs and outputs of the selected workflow. This can be edited and updated as described in the "Running an AnVIL workflow within R" vigenette.

Examples

```
## Not run:
workspace()
browse_workspace(use_avworkspace = FALSE)
tbl <- table()
wkflw <- avworkflow_gadget()

## End(Not run)
```

`drs_stat` *DRS (Data Repository Service) URL management*

Description

`drs_stat()` resolves zero or more DRS URLs to their google bucket location.

`drs_access_url()` returns a vector of 'signed' URLs that allow access to restricted resources via standard https protocols.

`drs_cp()` copies 0 or more DRS URIs to a google bucket or local folder

Usage

```
drs_stat(source = character(), region = "US")

drs_access_url(source = character(), region = "US")

drs_cp(source, destination, ..., overwrite = FALSE)
```

Arguments

`source`	character() DRS URLs (beginning with 'drs://') to resources managed by the 'martha' DRS resolution server.
`region`	character(1) Google cloud 'region' in which the DRS resource is located. Most data is located in `"US"` (the default); in principle `"auto"` allows for discovery of the region, but sometimes fails. Regions are enumerated at `https://cloud.google.com/storage/docs/locations#available-locations`.

`destination`	`character(1)`, google cloud bucket or local file system destination path.
`...`	additional arguments, passed to `gsutil_cp()` for file copying.
`overwrite`	logical(1) indicating that source `fileNames` present in `destination` should downloaded again.

Details

`drs_stat()` sends requests in parallel to the DRS server, using 8 forked processes (by default) to speed up queries. Use `options(mc.cores = 16L)`, for instance, to set the number of processes to use.

`drs_stat()` uses the AnVIL 'pet' account associated with a runtime. The pet account is discovered by default when evaluated on an AnVIL runtime (e.g., in RStudio or a Jupyter notebook in the AnVIL), or can be found in the return value of `avruntimes()`.

Errors reported by the DRS service are communicated to the user, but can be cryptic. The DRS service itself is called 'martha'. Errors mentioning martha might commonly involve a mal-formed DRS uri. Martha uses a service called 'bond' to establish credentials with registered third party entities such as Gen3 or Kids First. Errors mentioning bond might involve absence of credentials, within Terra, to access the resource; check that, in the Terra / AnVIL graphical user interface, the user profiles 'External Entities' includes the organization to which the DRS uri is being resolved.

Value

`drs_stat()` returns a tbl with the following columns:

- fileName: character() (resolver sometimes returns null).
- size: integer() (resolver sometimes returns null).
- contentType: character() (resolver sometimes returns null).
- gsUri: character() (resolver sometimes returns null).
- timeCreated: character() (the time created formatted using ISO 8601; resolver sometimes returns null).
- timeUpdated: character() (the time updated formatted using ISO 8601; resolver sometimes returns null).
- bucket: character() (resolver sometimes returns null).
- name: character() (resolver sometimes returns null).
- googleServiceAccount: list() (null unless the DOS url belongs to a Bond supported host).
- hashes: list() (contains the hashes type and their checksum value; if unknown. it returns null)

`drs_access_url()` returns a vector of https URLs corresponding to the vector of DRS URIs provided as inputs to the function.

`drs_cp()` returns a tibble like `drs_stat()`, but with additional columns

- simple: logical() value indicating whether resolution used a simple signed URL (`TRUE`) or auxilliary service account.
- destination: character() full path to retrieved object(s)

Examples

```
drs <- c(
    vcf = "drs://dg.ANV0/6f633518-f2de-4460-aaa4-a27ee6138ab5",
    tbi = "drs://dg.ANV0/4fb9e77f-c92a-4deb-ac90-db007dc633aa"
)

if (gcloud_exists() && startsWith(gcloud_account(), "pet-")) {
    ## from within AnVIL
    tbl <- drs_stat(uri)
    urls <- drs_access_url(uri)
    ## library(VariantAnnotation)
    ## vcffile <- VcfFile(urls[["vcf"]], urls[["tbi"]])
    ##
    ## header <- scanVcfHeader(vcffile)
    ## meta(header)[["contig"]]
}
```

`gcloud` *gcloud command line utility interface*

Description

These functions invoke the `gcloud` command line utility. See gsutil for details on how `gcloud` is located.

`gcloud_exists()` tests whether the `gcloud()` command can be found on this system. See 'Details' section of `gsutil` for where the application is searched.

`gcloud_account()`: report the current gcloud account via `gcloud config get-value account`.

`gcloud_project()`: report the current gcloud project via `gcloud config get-value project`.

`gcloud_help()`: queries `gcloud` for help for a command or sub-comand via `gcloud help ....`

`gcloud_cmd()` allows arbitrary `gcloud` command execution via `gcloud ....` Use pre-defined functions in preference to this.

Usage

```
gcloud_exists()

gcloud_account(account = NULL)

gcloud_project(project = NULL)

gcloud_help(...)

gcloud_cmd(cmd, ...)
```

Arguments

`account`	character(1) Google account (e.g., `user@gmail.com`) to use for authentication.
`project`	character(1) billing project name.
`...`	Additional arguments appended to gcloud commands.
`cmd`	`character(1)` representing a command used to evaluate `gcloud cmd ...`.

Value

`gcloud_exists()` returns `TRUE` when the `gcloud` application can be found, FALSE otherwise.

`gcloud_account()` returns a `character(1)` vector containing the active gcloud account, typically a gmail email address.

`gcloud_project()` returns a `character(1)` vector containing the active gcloud project.

`gcloud_help()` returns an unquoted `character()` vector representing the text of the help manual page returned by `gcloud help ...`.

`gcloud_cmd()` returns a `character()` vector representing the text of the output of `gcloud cmd ...`

Examples

```
gcloud_exists()

if (gcloud_exists())
    gcloud_account()

if (gcloud_exists())
    gcloud_help()
```

`gsutil` *gsutil command line utility interface*

Description

These functions invoke the `gsutil` command line utility. See the "Details:" section if you have gsutil installed but the package cannot find it.

`gsutil_requesterpays()`: does the google bucket require that the requester pay for access?

`gsutil_ls()`: List contents of a google cloud bucket or, if `source` is missing, all Cloud Storage buckets under your default project ID

`gsutil_exists()`: check if the bucket or object exists.

`gsutil_stat()`: print, as a side effect, the status of a bucket, directory, or file.

`gsutil_cp()`: copy contents of `source` to `destination`. At least one of `source` or `destination` must be Google cloud bucket; `source` can be a character vector with length greater than 1. Use `gsutil_help("cp")` for `gsutil` help.

`gsutil_rm()`: remove contents of a google cloud bucket.

`gsutil_rsync()`: synchronize a source and a destination. If the destination is on the local file system, it must be a directory or not yet exist (in which case a directory will be created).

`gsutil_cat()`: concatenate bucket objects to standard output

`gsutil_help()`: print 'man' page for the `gsutil` command or subcommand. Note that only commandes documented on this R help page are supported.

`gsutil_pipe()`: create a pipe to read from or write to a gooogle bucket object.

Usage

```
gsutil_requesterpays(source)

gsutil_ls(source = character(), ..., recursive = FALSE)

gsutil_exists(source)

gsutil_stat(source)

gsutil_cp(source, destination, ..., recursive = FALSE, parallel = TRUE)

gsutil_rm(source, ..., force = FALSE, recursive = FALSE, parallel = TRUE)

gsutil_rsync(
  source,
  destination,
  ...,
  exclude = NULL,
  dry = TRUE,
  delete = FALSE,
  recursive = FALSE,
  parallel = TRUE
)

gsutil_cat(source, ..., header = FALSE, range = integer())

gsutil_help(cmd = character(0))

gsutil_pipe(source, open = "r", ...)
```

Arguments

`source`	`character(1)`, (`character()` for `gsutil_requesterpays()`, `gsutil_ls()`, `gsutil_exists()`, `gsutil_cp()`) paths to a google storage bucket, possibly with wild-cards for file-level pattern matching.
`...`	additional arguments passed as-is to the `gsutil` subcommand.
`recursive`	`logical(1)`; perform operation recursively from `source`?. Default: `FALSE`.
`destination`	`character(1)`, google cloud bucket or local file system destination path.
`parallel`	`logical(1)`, perform parallel multi-threaded / multi-processing (default is `TRUE`).

force	`logical(1)`: continue silently despite errors when removing multiple objects. Default: `FALSE`.
exclude	`character(1)` a python regular expression of bucket paths to exclude from synchronization. E.g., `'.*(\\.png|\\.txt)$"` excludes '.png' and .txt' files.
dry	`logical(1)`, when `TRUE` (default), return the consequences of the operation without actually performing the operation.
delete	`logical(1)`, when `TRUE`, remove files in `destination` that are not in `source`. Exercise caution when you use this option: it's possible to delete large amounts of data accidentally if, for example, you erroneously reverse source and destination.
header	`logical(1)` when `TRUE` annotate each
range	(optional) `integer(2)` vector used to form a range from-to of bytes to concatenate. `NA` values signify concatenation from the start (first position) or to the end (second position) of the file.
cmd	`character()` (optional) command name, e.g., `"ls"` for help.
open	`character(1)` either `"r"` (read) or `"w"` (write) from the bucket.

Details

The `gsutil` system command is required. The search for `gsutil` starts with environment variable `GCLOUD_SDK_PATH` providing a path to a directory containing a `bin` directory containingin `gsutil`, `gcloud`, etc. The path variable is searched for first as an `option()` and then system variable. If no option or global variable is found, `Sys.which()` is tried. If that fails, `gsutil` is searched for on defined paths. On Windows, the search tries to find `Google\\Cloud SDK\\google-cloud-sdk\\bin\\gsutil.cmd` in the `LOCAL APP DATA`, `Program Files`, and `Program Files (x86)` directories. On linux / macOS, the search continues with `~/google-cloud-sdk`.
`gsutil_rsync()': To make `"gs://mybucket/data"`match the contents of the local directory`"data"‘ you could do:

`gsutil_rsync("data", "gs://mybucket/data", delete = TRUE)`

To make the local directory "data" the same as the contents of gs://mybucket/data:

`gsutil_rsync("gs://mybucket/data", "data", delete = TRUE)`

If `destination` is a local path and does not exist, it will be created.

Value

`gsutil_requesterpays()`: named `logical()` vector TRUE when requester-pays is enabled.

`gsutil_ls()`: `character()` listing of `source` content.

`gsutil_exists()`: logical(1) TRUE if bucket or object exists.

`gsutil_stat()`: `tibble()` summarizing status of each bucket member.

`gsutil_cp()`: exit status of `gsutil_cp()`, invisibly.

`gsutil_rm()`: exit status of `gsutil_rm()`, invisibly.

`gsutil_rsync()`: exit status of `gsutil_rsync()`, invisbly.

`gsutil_cat()` returns the content as a character vector.

`gsutil_help()`: `character()` help text for subcommand `cmd`.

`gsutil_pipe()` an unopened R `pipe()`; the mode is *not* specified, and the pipe must be used in the appropriate context (e.g., a pipe created with `open = "r"` for input as `read.csv()`)

Examples

```
    src <- "gs://genomics-public-data/1000-genomes/other/sample_info/sample_info.csv"
if (gcloud_exists())
    gsutil_requesterpays(src) # FALSE -- no cost download

if (gcloud_exists()) {
    gsutil_exists(src)
    gsutil_stat(src)
    gsutil_ls(dirname(src))
}

if (gcloud_exists()) {
   gsutil_cp(src, tempdir())
   ## gsutil_*() commands work with spaces in the source or destination
   destination <- file.path(tempdir(), "foo bar")
   gsutil_cp(src, destination)
   file.exists(destination)
}

if (gcloud_exists())
    gsutil_help("ls")

if (gcloud_exists()) {
    df <- read.csv(gsutil_pipe(src), 5L)
    class(df)
    dim(df)
    head(df)
}
```

`install` *Discover binary packages for fast installation*

Description

`install()` is deprecated in favor of `BiocManager::install()`.

`repository()` is deprecated in favor of `BiocManager::containerRepository()`.

`repositories()` is deprecated in favor of `BiocManager::repositories()`.
`repository_stats()`: summarize binary packages compatible with the Bioconductor or Terra container in use.

`add_libpaths()`: Add local library paths to `.libPaths()`.

Usage

```
install(
  pkgs = character(),
  ...,
  version = BiocManager::version(),
  binary_base_url = BINARY_BASE_URL
)

repository(version = BiocManager::version(), binary_base_url = BINARY_BASE_URL)

repositories(
  version = BiocManager::version(),
  binary_base_url = BINARY_BASE_URL
)

repository_stats(
  version = BiocManager::version(),
  binary_base_url = BINARY_BASE_URL
)

## S3 method for class 'repository_stats'
print(x, ...)

add_libpaths(paths)
```

Arguments

`pkgs`	`character()` packages to install from binary repository.
`...`	additional arguments. `install()` passes additional arguments to `BiocManager::install()`. `print.repository_stats()` ignores the additional arguments.
`version`	`character(1)` or `package_version` Bioconductor version, e.g., "3.12".
`binary_base_url`	`character(1)` host and base path for binary package 'CRAN-style' repository; not usually required by the end-user.
`x`	the object returned by `repository_stats()`.
`paths`	`character()`: vector of directories to add to `.libPaths()`. Paths that do not exist will be created.

Value

`repository_stats()` returns a list of class `repository_stats` with the following fields:

- container: character(1) container label, e.g., `bioconductor_docker`, or NA if not evaluated on a supported container
- bioconductor_version: `package_version` the Bioconductor version provided by the user.
- repository_exists: logical(1) TRUE if a binary repository exists for the container and Bioconductor_Version version.

- bioconductor_binary_repository: character(1) repository location, if available, or NA if the repository does not exist.
- n_software_packages: integer(1) number of software packages in the Bioconductor source repository.
- n_binary_packages: integer(1) number of binary packages available. When a binary repository exists, this number is likely to be larger than the number of source software packages, because it includes the binary version of the source software packages, as well as the (possibly CRAN) dependencies of the binary packages
- n_binary_software_packages: integer(1) number of binary packages derived from Bioconductor source packages. This number is less than or equal to `n_software_packages`.
- missing_binaries: integer(1) the number of Bioconductor source software packages that are not present in the binary repository.
- out_of_date_binaries: integer(1) the number of Bioconductor source software packages that are newer than their binary counterpart. A newer source software package might occur when the main Bioconductor build system has updated a package after the most recent run of the binary build system.

`add_libpaths()`: updated .libPaths(), invisibly.

Functions

- `print(repository_stats)`: Print a summary of package availability in binary repositories.

Examples

```
stats <- repository_stats() # obtain statistics
stats                       # display a summary
stats$container             # access an element for further computation

## Not run: add_libpaths("/tmp/host-site-library")
```

`localize` *Copy packages, folders, or files to or from google buckets.*

Description

`localize()`: recursively synchronizes files from a Google storage bucket (`source`) to the local file system (`destination`). This command acts recursively on the `source` directory, and does not delete files in `destination` that are not in ‘source.

`delocalize()`: synchronize files from a local file system (`source`) to a Google storage bucket (`destination`). This command acts recursively on the `source` directory, and does not delete files in `destination` that are not in `source`.

Usage

```
localize(source, destination, dry = TRUE)

delocalize(source, destination, unlink = FALSE, dry = TRUE)
```

Arguments

`source`	`character(1)`, a google storage bucket or local file system directory location.
`destination`	`character(1)`, a google storage bucket or local file system directory location.
`dry`	`logical(1)`, when `TRUE` (default), return the consequences of the operation without actually performing the operation.
`unlink`	`logical(1)` remove (unlink) the file or directory in `source`. Default: `FALSE`.

Value

`localize()`: exit status of function `gsutil_rsync()`.

`delocalize()`: exit status of function `gsutil_rsync()`

`Response` *Process service responses to tibble and other data structures.*

Description

Process service responses to tibble and other data structures.

Usage

```
flatten(x)

## S4 method for signature 'response'
str(object)

## S3 method for class 'response'
as.list(x, ..., as = c("text", "raw", "parsed"))
```

Arguments

`x`	A `response` object returned by the service.
`object`	A `response` object returned by the service.
`...`	not currently used
`as`	character(1); one of 'raw', 'text', 'parsed'

Value

`flatten()` returns a `tibble` where each row correseponds to a top-level list element of the return value, and columns are the unlisted second and more-nested elements.

`str()` displays a compact representation of the list-like JSON response; it returns `NULL`.

`as.list()` retruns the content of the web service request as a list.

Examples

```
if (gcloud_exists()) {
    leonardo <- Leonardo()
    leonardo$listRuntimes() %>% flatten()
}

if (gcloud_exists())
   leonardo$getSystemStatus() %>% str()

if (gcloud_exists())
    leonardo$getSystemStatus() %>% as.list()
```

`Service` *RESTful service constructor*

Description

RESTful service constructor

Usage

```
Service(
  service,
  host,
  config = httr::config(),
  authenticate = TRUE,
  api_url = character(),
  package = "AnVIL",
  schemes = "https",
  api_reference_url = api_url,
  api_reference_md5sum = character(),
  api_reference_headers = NULL
)
```

Arguments

`service`	character(1) The `Service` class name, e.g., `"terra"`.
`host`	character(1) host name that provides the API resource, e.g., `"api.firecloud.org"`.
`config`	httr::config() curl options

`authenticate`	logical(1) use credentials from authentication service file 'auth.json' in the specified package?
`api_url`	optional character(1) url location of OpenAPI `.json` or `.yaml` service definition.
`package`	character(1) (default `AnVIL`) The package where 'api.json' yaml and (optionally) 'auth.json' files are located.
`schemes`	character(1) (default 'https') Specifies the transfer protocol supported by the API service.
`api_reference_url`	character(1) path to reference API. See Details.
`api_reference_md5sum`	character(1) the result of `tools::md5sum()` applied to the reference API.
`api_reference_headers`	character() header(s) to be used (e.g., `c(Authorization = paste("Bearer", token))`) when retrieving the API reference for validation.

Details

This function creates a RESTful interface to a service provided by a host, e.g., "api.firecloud.org". The function requires an OpenAPI `.json` or `.yaml` specifcation as well as an (optional) `.json` authentication token. These files are located in the source directory of a pacakge, at `<package>/inst/service/<service>/ap` and `<package>/inst/service/<service>/auth.json`, or at `api_url`.

When provided, the `api_reference_md5sum` is used to check that the file described at `api_reference_url` has the same checksum as an author-validated version.

The service is usually a singleton, created at the package level during `.onLoad()`.

Value

An object of class `Service`.

Examples

```
.MyService <- setClass("MyService", contains = "Service")

MyService <- function() {
    .MyService(Service("my_service", host="my.api.org"))
}
```

`Services` *RESTful services useful for AnVIL developers*

Description

RESTful services useful for AnVIL developers

Usage

```
empty_object

operations(x, ..., .deprecated = FALSE)

## S4 method for signature 'Service'
operations(x, ..., auto_unbox = FALSE, .deprecated = FALSE)

schemas(x)

tags(x, .tags, .deprecated = FALSE)

## S4 method for signature 'Service'
x$name

Leonardo()

Terra()

Rawls()

Dockstore()

Gen3Fence()

Gen3Indexd()

Gen3Sheepdog()

Gen3Peregrine()
```

Arguments

`x`	A `Service` instance, usually a singleton provided by the package and documented on this page, e.g., `leonardo` or `terra`.
`...`	additional arguments passed to methods or, for `operations,Service-method`, to the internal `get_operation()` function.
`.deprecated`	optional logical(1) include deprecated operations?
`auto_unbox`	logical(1) If FALSE (default) do not automatically 'unbox' R scalar values from JSON arrays to JSON scalers.
`.tags`	optional character() of tags to use to filter operations.
`name`	A symbol representing a defined operation, e.g., `leonardo$listRuntimes()`.

Details

When using `$` to select a service, some arguments appear in 'body' of the REST request. Specify these using the `.__body__`= argument, as illustrated for `createBillingProjectFull()`, below.

Value

`empty_object` returns a representation to be used as arguments in function calls expecting the empty json object \{\}.

`Leonardo()` creates the API of the Leonard container deployment service at https://notebooks.firecloud.org/api-docs.yaml.

`Terra()` creates the API of the Terra cloud computational environemnt at https://api.firecloud.org/.

`Rawls()` creates the API of the Rawls cloud computational environemnt at https://rawls.dsde-prod.broadinstitute.org.

`Dockstore()` represents the API of the Dockstore platform to share Docker-based tools in CWL or WDL or Nextflow at https://dockstore.org
`gen3_*` APIs are not fully implemented, because a service endpoint has not been identified.

`Gen3Fence()` returns the authentication API at https://raw.githubusercontent.com/uc-cdis/fence/master/openapis/swagger.yar

`Gen3Indexd()` returns the indexing service API documented at https://raw.githubusercontent.com/uc-cdis/indexd/master/openapis/swagger.yaml

`Gen3Sheepdog` returns the submission services API at https://raw.githubusercontent.com/uc-cdis/sheepdog/master/openapi/sv

`Gen3Peregrine` returns the graphQL query services API at https://raw.githubusercontent.com/uc-cdis/peregrine/master/openapis/swagger.yaml

Examples

```
empty_object

if (gcloud_exists()) {
    ## Arguments to be used as the 'body' (`.__body__=`) of a REST query
    Terra()$createBillingProjectFull       # 6 arguments...
    args(Terra()$createBillingProjectFull) # ... passed as `.__body__ = list(...)`
}
if (gcloud_exists())
    Leonardo()

if (gcloud_exists()) {
    tags(Terra())
    tags(Terra(), "Billing")
}

if (gcloud_exists()) {
    tags(Rawls())
    tags(Rawls(), "billing")
}

Dockstore()
```

Index