Package 'connectapi'

January 31, 2023

Type Package

Title Utilities for Interacting with the 'Posit Connect' Server API

Version 0.1.3.1

Description Provides a helpful 'R6' class and methods for interacting with
the 'Posit Connect' Server API along with some meaningful utility functions
for regular tasks. API documentation varies by 'Posit Connect' installation
and version, but the latest documentation is also hosted publicly at
<https://docs.posit.co/connect/api/>.

License MIT + file LICENSE

URL https://github.com/rstudio/connectapi

Imports config,
dplyr,
fs,
glue,
httr,
lifecycle,
magrittr,
progress,
purrr,
R6,
rlang (>= 0.4.2),
tibble,
uuid,
vctrs (>= 0.3.0),
yaml,
bit64,
jsonlite

Suggests crayon,
flexdashboard,
ggplot2,
gridExtra,
htmltools,
knitr,
lubridate,
processx,
rmarkdown,
rprojroot,
spelling,

testthat,
rsconnect,
webshot2,
covr,
mockery,
devtools

VignetteBuilder knitr

RdMacros lifecycle

Encoding UTF-8

Language en-US

RoxygenNote 7.2.3

Roxygen list(markdown = TRUE)

R topics documented:

`connectapi-package` *connectapi: Utilities for Interacting with the 'Posit Connect' Server API*

Description

Provides a helpful 'R6' class and methods for interacting with the 'Posit Connect' Server API along with some meaningful utility functions for regular tasks. API documentation varies by 'Posit Connect' installation and version, but the latest documentation is also hosted publicly at `https://docs.posit.co/connect/api/`.

Author(s)

Maintainer: Cole Arendt <cole@posit.co>

Authors:

- Sean Lopp

Other contributors:

- Posit, PBC [copyright holder, funder]

See Also

Useful links:

- https://github.com/rstudio/connectapi

`acl_add_user` *ACL Add Users or Groups*

Description

Add a user or group to the content as an "owner" (collaborator) or "viewer"

Usage

```
acl_add_user(content, user_guid, role)

acl_add_group(content, group_guid, role)

acl_add_collaborator(content, user_guid)

acl_add_viewer(content, user_guid)

acl_remove_user(content, user_guid)

acl_add_self(content)

acl_remove_self(content)

acl_remove_group(content, group_guid)
```

Arguments

`content`	The R6 Content object (as returned by `content_item()`)
`user_guid`	The user's GUID. Use `get_users()`
`role`	One of "owner" or "viewer"
`group_guid`	The group's GUID. Use `get_groups()`

Details

- `acl_add_user()` allows you to add ACL for a user and specify role
- `acl_add_group()` allows you to add ACL for a group and specify role
- `acl_add_collaborator()` is a helper to add a `user` collaborators
- `acl_add_viewer()` is a helper to add a `user` viewer
- `acl_remove_user()` removes a user's ACLs from a piece of content
- `acl_remove_group()` removes a group's ACLs from a piece of content
- `acl_add_self()` is useful for admins and adds the current user as a collaborator
- `acl_remove_self()` removes the current user's ACLs from a piece of content

Value

The R6 content object (for piping)

See Also

Other content functions: `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url_chr()`, `dashboard_url()`, `delete_vanity_url()`, `deploy_repo()`, `get_acl_user()`, `get_bundles()`, `get_environment()`, `get_image()`, `get_jobs()`, `get_vanity_url()`, `git`, `permissions`, `set_image_path()`, `set_run_as()`, `set_vanity_url()`, `swap_vanity_url()`, `verify_content_name()`

`audit_access_open` *Audit Access Controls*

Description

[Experimental]

Usage

```
audit_access_open(apps, type = "all")
```

Arguments

`apps`	App list, see `cache_apps`
`type`	One of "all" or "logged_in". If "all", return a list of apps whose access control is set to "Everyone". If "logged_in", return a list of apps whose access control is set to "All logged in users"

See Also

Other audit functions: `audit_r_versions()`, `audit_runas()`, `audit_vanity_urls()`, `cache_apps()`

audit_runas *Audit Run As Settings*

Description

[Experimental]

Usage

```
audit_runas(apps)
```

Arguments

apps	App list, see cache_apps

Value

A data frame with the app name and the Run As user if the Run As user is not the default

See Also

Other audit functions: audit_access_open(), audit_r_versions(), audit_vanity_urls(), cache_apps()

audit_r_versions *Audit R Versions*

Description

[Experimental]

Usage

```
audit_r_versions(apps)
```

Arguments

apps	App list, see cache_apps

Value

A plot that shows the R version used by content over time and in aggregate.

See Also

Other audit functions: audit_access_open(), audit_runas(), audit_vanity_urls(), cache_apps()

`audit_vanity_urls` *Audit Vanity URLs*

Description

[Experimental]

Usage

```
audit_vanity_urls(apps, server_url, vanity = NULL)
```

Arguments

`apps`	App list, see `cache_apps()`
`server_url`	Base url for the Connect server
`vanity`	Optional, see details

Details

If `vanity` is not provided, returns a list of all the vanity urls in use. If `vanity` is provided, returns whether or not `vanity` is eligible as a vanity url.

See Also

Other audit functions: `audit_access_open()`, `audit_r_versions()`, `audit_runas()`, `cache_apps()`

`browse_solo` *Browse*

Description

Browse to different locations on Connect via utils::browseURL

Usage

```
browse_solo(content)

browse_dashboard(content)

browse_api_docs(connect)

browse_connect(connect)
```

Arguments

`content`	A R6 Content object
`connect`	A R6 Connect object

Value

The url that is opened in the browser

Bundle *Bundle*

Description

An R6 class that represents a bundle

Methods

Public methods:

- `Bundle$new()`
- `Bundle$print()`
- `Bundle$clone()`

Method `new()`:

Usage:

```
Bundle$new(path)
```

Method `print()`:

Usage:

```
Bundle$print(...)
```

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

Usage:

```
Bundle$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other R6 classes: `ContentTask`, `Content`, `Environment`, `PositConnect`, `Task`, `Vanity`, `VariantSchedule`, `VariantTask`, `Variant`

bundle_dir *Define a bundle from a Directory*

Description

Creates a bundle from a target directory.

Usage

```
bundle_dir(
  path = ".",
  filename = fs::file_temp(pattern = "bundle", ext = ".tar.gz")
)
```

Arguments

`path`	The path to the directory to be bundled
`filename`	The output bundle path

Value

Bundle A bundle object

See Also

Other deployment functions: `bundle_path()`, `bundle_static()`, `deploy()`, `download_bundle()`, `poll_task()`

`bundle_path` *Define a bundle from a path (a path directly to a tar.gz file)*

Description

Define a bundle from a path (a path directly to a tar.gz file)

Usage

```
bundle_path(path)
```

Arguments

`path`	The path to a .tar.gz file

Value

Bundle A bundle object

See Also

Other deployment functions: `bundle_dir()`, `bundle_static()`, `deploy()`, `download_bundle()`, `poll_task()`

`bundle_static` *Define a bundle from a static file (or files)*

Description

Defines a bundle from static files. It copies all files to a temporary directory, generates a basic manifest file (using the first file as the "primary"), and bundles the directory.

Usage

```
bundle_static(
  path,
  filename = fs::file_temp(pattern = "bundle", ext = ".tar.gz")
)
```

Arguments

`path`	The path to a file (or files) that will be used for the static bundle
`filename`	The output bundle path

Details

NOTE: the `rsconnect` package is required for this function to work properly.

Value

Bundle A bundle object

See Also

Other deployment functions: `bundle_dir()`, `bundle_path()`, `deploy()`, `download_bundle()`, `poll_task()`

`cache_apps` *Get information on all apps for a server*

Description

[Experimental]

Usage

```
cache_apps(connect)
```

Arguments

`connect`	A Connect object

Value

List with application data, to be used by audit functions

See Also

Other audit functions: `audit_access_open()`, `audit_r_versions()`, `audit_runas()`, `audit_vanity_urls()`

`connect` *Create a connection to Posit Connect*

Description

Creates a connection to Posit Connect using the server URL and an api key. Validates the connection and checks that the version of the server is compatible with the current version of the package.

Usage

```
connect(
  server = Sys.getenv(paste0(prefix, "_SERVER"), NA_character_),
  api_key = Sys.getenv(paste0(prefix, "_API_KEY"), NA_character_),
  prefix = "CONNECT",
  ...,
  .check_is_fatal = TRUE
)
```

Arguments

`server`	The URL for accessing Posit Connect. Defaults to environment variable CONNECT_SERVER
`api_key`	The API Key to authenticate to Posit Connect with. Defaults to environment variable CONNECT_API_KEY
`prefix`	The prefix used to determine environment variables
`...`	Additional arguments. Not used at present
`.check_is_fatal`	Whether to fail if "check" requests fail. Useful in rare cases where more http request customization is needed for requests to succeed.

Value

A Posit Connect R6 object that can be passed along to methods

Examples

```
## Not run:
connect()

## End(Not run)


# default is to read CONNECT_SERVER and CONNECT_API_KEY environment variables
# this example will read TEST_1_SERVER and TEST_1_API_KEY
connect(prefix = "TEST_1")
```

`Content` *Content*

Description

An R6 class that represents content

Public fields

`connect`	An R6 Connect object
`content`	The content details from Posit Connect

Methods

Public methods:

- `Content$new()`
- `Content$get_connect()`
- `Content$get_content()`
- `Content$get_content_remote()`
- `Content$get_bundles()`
- `Content$bundle_download()`
- `Content$bundle_delete()`
- `Content$internal_content()`
- `Content$update()`
- `Content$danger_delete()`
- `Content$runas()`
- `Content$get_url()`
- `Content$get_dashboard_url()`
- `Content$get_jobs()`
- `Content$get_job()`
- `Content$jobs()`
- `Content$job()`
- `Content$variants()`
- `Content$tag_set()`
- `Content$tag_delete()`
- `Content$tags()`
- `Content$permissions_add()`
- `Content$permissions_update()`
- `Content$permissions_delete()`
- `Content$permissions()`
- `Content$environment()`
- `Content$environment_set()`
- `Content$environment_all()`
- `Content$deploy()`
- `Content$repo_enable()`
- `Content$repo_set()`

- Content$print()
- Content$clone()

Method `new()`**:**

Usage:

```
Content$new(connect, content)
```

Method `get_connect()`**:**

Usage:

```
Content$get_connect()
```

Method `get_content()`**:**

Usage:

```
Content$get_content()
```

Method `get_content_remote()`**:**

Usage:

```
Content$get_content_remote()
```

Method `get_bundles()`**:**

Usage:

```
Content$get_bundles()
```

Method `bundle_download()`**:**

Usage:

```
Content$bundle_download(
  bundle_id,
  filename = tempfile(pattern = "bundle", fileext = ".tar.gz"),
  overwrite = FALSE
)
```

Method `bundle_delete()`**:**

Usage:

```
Content$bundle_delete(bundle_id)
```

Method `internal_content()`**:**

Usage:

```
Content$internal_content()
```

Method `update()`**:**

Usage:

```
Content$update(...)
```

Method `danger_delete()`**:**

Usage:

```
Content$danger_delete()
```

Method `runas()`**:**

Usage:

```
Content$runas(run_as, run_as_current_user = FALSE)
```

Method `get_url()`:

Usage:

```
Content$get_url()
```

Method `get_dashboard_url()`:

Usage:

```
Content$get_dashboard_url(pane = "")
```

Method `get_jobs()`:

Usage:

```
Content$get_jobs()
```

Method `get_job()`:

Usage:

```
Content$get_job(key)
```

Method `jobs()`:

Usage:

```
Content$jobs()
```

Method `job()`:

Usage:

```
Content$job(key)
```

Method `variants()`:

Usage:

```
Content$variants()
```

Method `tag_set()`:

Usage:

```
Content$tag_set(tag_id)
```

Method `tag_delete()`:

Usage:

```
Content$tag_delete(id)
```

Method `tags()`:

Usage:

```
Content$tags()
```

Method `permissions_add()`:

Usage:

```
Content$permissions_add(principal_guid, principal_type, role)
```

Method `permissions_update()`:

Usage:

```
Content$permissions_update(id, principal_guid, principal_type, role)
```

Method `permissions_delete()`:

Usage:

```
Content$permissions_delete(id)
```

Method `permissions()`**:**

Usage:

```
Content$permissions(id = NULL, add_owner = FALSE)
```

Method `environment()`**:**

Usage:

```
Content$environment()
```

Method `environment_set()`**:**

Usage:

```
Content$environment_set(...)
```

Method `environment_all()`**:**

Usage:

```
Content$environment_all(...)
```

Method `deploy()`**:**

Usage:

```
Content$deploy(bundle_id = NULL)
```

Method `repo_enable()`**:**

Usage:

```
Content$repo_enable(enabled = TRUE)
```

Method `repo_set()`**:**

Usage:

```
Content$repo_set(repository, branch, subdirectory)
```

Method `print()`**:**

Usage:

```
Content$print(...)
```

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

Usage:

```
Content$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other R6 classes: `Bundle`, `ContentTask`, `Environment`, `PositConnect`, `Task`, `Vanity`, `VariantSchedule`, `VariantTask`, `Variant`

ContentTask *ContentTask*

Description

An R6 class that represents a Task for a piece of Content

Super class

`connectapi::Content` -> `ContentTask`

Methods

Public methods:

- `ContentTask$new()`
- `ContentTask$get_task()`
- `ContentTask$add_data()`
- `ContentTask$get_data()`
- `ContentTask$print()`
- `ContentTask$clone()`

Method `new()`**:**

Usage:

```
ContentTask$new(connect, content, task)
```

Method `get_task()`**:**

Usage:

```
ContentTask$get_task()
```

Method `add_data()`**:**

Usage:

```
ContentTask$add_data(data)
```

Method `get_data()`**:**

Usage:

```
ContentTask$get_data()
```

Method `print()`**:**

Usage:

```
ContentTask$print(...)
```

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

Usage:

```
ContentTask$clone(deep = FALSE)
```

Arguments:

`deep`	Whether to make a deep clone.

See Also

Other R6 classes: `Bundle`, `Content`, `Environment`, `PositConnect`, `Task`, `Vanity`, `VariantSchedule`, `VariantTask`, `Variant`

`content_delete` *Delete Content*

Description

Delete a content item. WARNING: This action deletes all history, configuration, logs, and resources about a content item. It *cannot* be undone.

Usage

```
content_delete(content, force = FALSE)
```

Arguments

`content`	an R6 content item
`force`	Optional. A boolean that determines whether we should prompt in interactive sessions

Value

The R6 Content item. The item is deleted, but information about it is cached locally

See Also

Other content functions: `acl_add_user()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url_chr()`, `dashboard_url()`, `delete_vanity_url()`, `deploy_repo()`, `get_acl_user()`, `get_bundles()`, `get_environment()`, `get_image()`, `get_jobs()`, `get_vanity_url()`, `git`, `permissions`, `set_image_path()`, `set_run_as()`, `set_vanity_url()`, `swap_vanity_url()`, `verify_content_name()`

`content_item` *Get Content Item*

Description

Returns a single content item based on guid

Usage

```
content_item(connect, guid)
```

Arguments

`connect`	A Connect object
`guid`	The GUID for the content item to be retrieved

Value

A Content object for use with other content endpoints

See Also

Other content functions: acl_add_user(), content_delete(), content_title(), content_update(), create_random_name(), dashboard_url_chr(), dashboard_url(), delete_vanity_url(), deploy_repo(), get_acl_user(), get_bundles(), get_environment(), get_image(), get_jobs(), get_vanity_url(), git, permissions, set_image_path(), set_run_as(), set_vanity_url(), swap_vanity_url(), verify_content_name()

Examples

```
## Not run:
  connect() %>%
    content_item("some-guid") %>%
    content_update_access_type("all")

## End(Not run)
```

content_list_by_tag *Content List*

Description

[Experimental] Get a content list

Usage

```
content_list_by_tag(src, tag)
```

Arguments

src	An R6 Connect object
tag	A connect_tag_tree object or tag ID

Details

content_list_by_tag() retrieves a content list by tag

content_list_with_permissions *Get Content List with Permissions*

Description

[Experimental] These functions are experimental placeholders until the API supports this behavior.

Usage

```
content_list_with_permissions(src, ..., .p = NULL)

content_list_guid_has_access(content_list, guid)
```

Arguments

`src`	A Connect R6 object
`...`	Extra arguments. Currently not used
`.p`	Optional. A predicate function, passed as-is to `purrr::keep()`. See `get_content()` for more details. Can greatly help performance by reducing how many items to get permissions for
`content_list`	A "content list with permissions" as returned by `content_list_with_permissions()`
`guid`	A user or group GUID to filter the content list by whether they have access

Details

`content_list_with_permissions` loops through content and retrieves permissions for each item (with a progress bar). This can take a long time for lots of content! Make sure to use the optional `.p` argument as a predicate function that filters the content list before it is transformed.

`content_list_guid_has_access` works with a `content_list_with_permissions` dataset by checking whether a given GUID (either user or group) has access to the content by:

- checking if the content has access_type == "all"
- checking if the content has access_type == "logged_in"
- checking if the provided guid is the content owner
- checking if the provided guid is in the list of content permissions (in the "permissions" column)

`content_title` *Get Content Title*

Description

Return content title for a piece of content. If the content is missing (deleted) or not visible, then returns the `default`

Usage

```
content_title(connect, guid, default = "Unknown Content")
```

Arguments

`connect`	A Connect object
`guid`	The GUID for the content item to be retrieved
`default`	The default value returned for missing or not visible content

Value

character. The title of the requested content

See Also

Other content functions: `acl_add_user()`, `content_delete()`, `content_item()`, `content_update()`, `create_random_name()`, `dashboard_url_chr()`, `dashboard_url()`, `delete_vanity_url()`, `deploy_repo()`, `get_acl_user()`, `get_bundles()`, `get_environment()`, `get_image()`, `get_jobs()`, `get_vanity_url()`, `git`, `permissions`, `set_image_path()`, `set_run_as()`, `set_vanity_url()`, `swap_vanity_url()`, `verify_content_name()`

content_update *Update Content*

Description

Update settings for a content item. For a list of all settings, see the latest documentation or the documentation for your server via `connectapi::browse_api_docs()`.

Usage

```
content_update(content, ...)

content_update_access_type(content, access_type = c("all", "logged_in", "acl"))

content_update_owner(content, owner_guid)
```

Arguments

`content`	An R6 content item
`...`	Settings up update that are passed along to Posit Connect
`access_type`	One of "all", "logged_in", or "acl"
`owner_guid`	The GUID of a user who is a publisher, so that they can become the new owner of the content

Details

Popular selections are `content_update(access_type="all")`, `content_update(access_type="logged_in")` or `content_update(access_type="acl")`, process settings, title, description, etc.

- `content_update_access_type()` is a helper to make it easier to change access_type
- `content_update_owner()` is a helper to make it easier to change owner

Value

An R6 content item

See Also

Other content functions: `acl_add_user()`, `content_delete()`, `content_item()`, `content_title()`, `create_random_name()`, `dashboard_url_chr()`, `dashboard_url()`, `delete_vanity_url()`, `deploy_repo()`, `get_acl_user()`, `get_bundles()`, `get_environment()`, `get_image()`, `get_jobs()`, `get_vanity_url()`, `git`, `permissions`, `set_image_path()`, `set_run_as()`, `set_vanity_url()`, `swap_vanity_url()`, `verify_content_name()`

`create_random_name` *Create Random Name*

Description

Creates a random name from the LETTERS dataset

Usage

```
create_random_name(length = 25)
```

Arguments

`length`	Optional. The length of the random name. Defaults to 25

Value

The randomly generated name

See Also

connectapi::verify_content_name

Other content functions: `acl_add_user()`, `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `dashboard_url_chr()`, `dashboard_url()`, `delete_vanity_url()`, `deploy_repo()`, `get_acl_user()`, `get_bundles()`, `get_environment()`, `get_image()`, `get_jobs()`, `get_vanity_url()`, `git`, `permissions`, `set_image_path()`, `set_run_as()`, `set_vanity_url()`, `swap_vanity_url()`, `verify_content_name()`

`dashboard_url` *Build a Dashboard URL from a Content Item*

Description

Returns the URL for the content dashboard (opened to the selected pane).

Usage

```
dashboard_url(content, pane = "")
```

Arguments

`content`	Content A Content object
`pane`	character The pane in the dashboard to link to

Value

character The dashboard URL for the content provided

See Also

Other content functions: `acl_add_user()`, `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url_chr()`, `delete_vanity_url()`, `deploy_repo()`, `get_acl_user()`, `get_bundles()`, `get_environment()`, `get_image()`, `get_jobs()`, `get_vanity_url()`, `git`, `permissions`, `set_image_path()`, `set_run_as()`, `set_vanity_url()`, `swap_vanity_url()`, `verify_content_name()`

`dashboard_url_chr` *Build a Dashboard URL from Character Vectors*

Description

Returns the URL for the content dashboard (opened to the selected pane). NOTE: this takes a character object for performance optimization.

Usage

```
dashboard_url_chr(connect_url, content_guid, pane = "")
```

Arguments

`connect_url`	character The base URL of the Connect server
`content_guid`	character The guid for the content item in question
`pane`	character The pane in the dashboard to link to

Value

character The dashboard URL for the content provided

See Also

Other content functions: `acl_add_user()`, `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url()`, `delete_vanity_url()`, `deploy_repo()`, `get_acl_user()`, `get_bundles()`, `get_environment()`, `get_image()`, `get_jobs()`, `get_vanity_url()`, `git`, `permissions`, `set_image_path()`, `set_run_as()`, `set_vanity_url()`, `swap_vanity_url()`, `verify_content_name()`

`delete_vanity_url` *Delete the Vanity URL*

Description

Deletes the Vanity URL for a piece of content.

Usage

```
delete_vanity_url(content)
```

Arguments

`content`	A Content object

See Also

Other content functions: `acl_add_user()`, `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url_chr()`, `dashboard_url()`, `deploy_repo()`, `get_acl_user()`, `get_bundles()`, `get_environment()`, `get_image()`, `get_jobs()`, `get_vanity_url()`, `git`, `permissions`, `set_image_path()`, `set_run_as()`, `set_vanity_url()`, `swap_vanity_url()`, `verify_content_name()`

`deploy` *Deploy a bundle*

Description

Deploys a bundle (tarball) to an Posit Connect server. If not provided, `name` (a unique identifier) will be an auto-generated alphabetic string. If deploying to an existing endpoint, you can set `name` or `guid` to the desired content.

Usage

```
deploy(
  connect,
  bundle,
  name = create_random_name(),
  title = name,
  guid = NULL,
  ...,
  .pre_deploy = {
 }
)

deploy_current(content)
```

Arguments

`connect`	A Connect object
`bundle`	A Bundle object
`name`	The unique name for the content on the server
`title`	optional The title to be used for the content on the server
`guid`	optional The GUID if the content already exists on the server
`...`	Additional arguments passed along to the content creation
`.pre_deploy`	An expression to execute before deploying the new bundle. The variables `content` and `bundle_id` are supplied
`content`	A Content object

Details

This function accepts the same arguments as `connectapi::content_update()`.

`deploy_current()` is a helper to easily redeploy the currently active bundle for an existing content item.

Value

Task A task object

See Also

connectapi::content_update

Other deployment functions: `bundle_dir()`, `bundle_path()`, `bundle_static()`, `download_bundle()`, `poll_task()`

Examples

```
## Not run:
  client <- connect()

  # beware bundling big directories, like `renv/`, `data/`, etc.
  bnd <- bundle_dir(".")

  deploy(client, bnd)

## End(Not run)
```

`deploy_repo` *Deploy a Git Repository*

Description

[Experimental] Deploy a git repository directly to Posit Connect, using Posit Connect's "pull-based" "git-polling" method of deployment.

Usage

```
deploy_repo(
  client,
  repository,
  branch,
  subdirectory,
  name = create_random_name(),
  title = name,
  ...
)

deploy_repo_enable(content, enabled = TRUE)

deploy_repo_update(content)
```

Arguments

`client`	A Connect R6 object
`repository`	The git repository to deploy
`branch`	The git branch to deploy
`subdirectory`	The subdirectory to deploy (must contain a `manifest.json`)
`name`	The "name" / unique identifier for the content. Defaults to a random character string
`title`	The "title" of the content
`...`	Additional options for defining / specifying content attributes
`content`	An R6 Content object (i.e. the result of `content_item()`)
`enabled`	Whether Connect will enable automatic polling for repository updates

Details

- `deploy_repo_enable()` enables (or disables) Posit Connect's git polling for a piece of content
- `deploy_repo_update()` triggers an update of the content from its git repository, if any are present

Value

A ContentTask object, for use with `poll_task()` (if you want to follow the logs)

See Also

connectapi::poll_task, connectapi::repo_check_branches, connectapi::repo_check_manifest_dirs

Other content functions: `acl_add_user()`, `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url_chr()`, `dashboard_url()`, `delete_vanity_url()`, `get_acl_user()`, `get_bundles()`, `get_environment()`, `get_image()`, `get_jobs()`, `get_vanity_url()`, `git`, `permissions`, `set_image_path()`, `set_run_as()`, `set_vanity_url()`, `swap_vanity_url()`, `verify_content_name()`

`download_bundle` *Download a Bundle from Deployed Connect Content*

Description

Downloads a Content item's active bundle, or (optionally) one of its other bundles.

Usage

```
download_bundle(
  content,
  filename = fs::file_temp(pattern = "bundle", ext = ".tar.gz"),
  bundle_id = NULL,
  overwrite = FALSE
)
```

Arguments

`content`	A Content object
`filename`	Optional. The output bundle path
`bundle_id`	Optional. A string representing the bundle_id to download. If NULL, will use the currently active bundle.
`overwrite`	Optional. Default FALSE. Whether to overwrite the target location if it already exists

Value

Bundle A bundle object

See Also

Other deployment functions: `bundle_dir()`, `bundle_path()`, `bundle_static()`, `deploy()`, `poll_task()`

`Environment` *Environment*

Description

An R6 class that represents a Content's Environment Variables

Super class

`connectapi::Content` -> `Environment`

Methods

Public methods:

- `Environment$new()`
- `Environment$environment()`
- `Environment$environment_set()`
- `Environment$environment_all()`
- `Environment$env_refresh()`
- `Environment$print()`
- `Environment$clone()`

Method `new()`**:**

Usage:

```
Environment$new(connect, content)
```

Method `environment()`**:**

Usage:

```
Environment$environment()
```

Method `environment_set()`**:**

Usage:

```
Environment$environment_set(...)
```

Method `environment_all()`**:**

Usage:

```
Environment$environment_all(...)
```

Method `env_refresh()`**:**

Usage:

```
Environment$env_refresh()
```

Method `print()`**:**

Usage:

```
Environment$print(...)
```

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

Usage:

```
Environment$clone(deep = FALSE)
```

Arguments:

`deep`	Whether to make a deep clone.

See Also

Other R6 classes: `Bundle`, `ContentTask`, `Content`, `PositConnect`, `Task`, `Vanity`, `VariantSchedule`, `VariantTask`, `Variant`

`get_acl_user` *Get ACL Details*

Description

[Deprecated] Retrieve the Access Controls associated with a given piece of content. Deprecated in favor of `content_add_user()` or permissions.

Usage

```
get_acl_user(content)

get_acl_group(content)

get_acl(content)

get_acl_user_role(content, user_guid)

get_acl_group_role(content, group_guid)
```

Arguments

`content`	Content An R6 Content item as returned from `content_item()`
`user_guid`	character A user guid. Get user guids using `get_users()`
`group_guid`	character A group guid. Get group guids using `get_groups()`

Details

NOTE: ACLs can still be stored, even when access_type for content is "all" or "logged_in" users. In these cases, granting or removing "viewer" privileges have no effect.

- `get_acl_user()` returns user ACLs
- `get_acl_group()` returns group ACLs
- `get_acl_user_role()` returns the "role" for a particular user on a piece of content
- `get_acl_group_role()` returns the "role" for a particular group on a piece of content

`get_acl()` is deprecated.

Value

A list of users/groups who have access to the content

See Also

Other content functions: `acl_add_user()`, `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url_chr()`, `dashboard_url()`, `delete_vanity_url()`, `deploy_repo()`, `get_bundles()`, `get_environment()`, `get_image()`, `get_jobs()`, `get_vanity_url()`, `git`, `permissions`, `set_image_path()`, `set_run_as()`, `set_vanity_url()`, `swap_vanity_url()`, `verify_content_name()`

`get_audit_logs` *Get Audit Logs from Posit Connect Server*

Description

Get Audit Logs from Posit Connect Server

Usage

```
get_audit_logs(src, limit = 20L, previous = NULL, nxt = NULL, asc_order = TRUE)
```

Arguments

`src`	The source object
`limit`	The number of records to return.
`previous`	Retrieve the previous page of Shiny application usage logs relative to the provided value. This value corresponds to an internal reference within the server and should be sourced from the appropriate attribute within the paging object of a previous response.
`nxt`	Retrieve the next page of Shiny application usage logs relative to the provided value. This value corresponds to an internal reference within the server and should be sourced from the appropriate attribute within the paging object of a previous response.
`asc_order`	Defaults to TRUE; Determines if the response records should be listed in ascending or descending order within the response. Ordering is by the started timestamp field.

Details

Please see https://docs.posit.co/connect/api/#getAuditLogs for more information

Value

A tibble with the following columns:

- **id**ID of the audit action
- **time**Timestamp in RFC3339 format when action was taken
- **user_id**User ID of the actor who made the audit action
- **user_description**Description of the actor
- **action**Audit action taken
- **event_description**Description of action

Examples

```
## Not run:
library(connectapi)
client <- connect()

# get the last 20 audit logs
get_audit_logs(client, limit = 20, asc_order = FALSE)

## End(Not run)
```

`get_bundles` *Get Bundles*

Description

Lists bundles for a content item

Usage

```
get_bundles(content, limit = Inf)

delete_bundle(content, bundle_id)
```

Arguments

`content`	A R6 Content item, as returned by `content_item()`
`limit`	Optional. Limit on number of bundles to return. Default Infinity.
`bundle_id`	A specific bundle ID for a content item

See Also

Other content functions: `acl_add_user()`, `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url_chr()`, `dashboard_url()`, `delete_vanity_url()`, `deploy_repo()`, `get_acl_user()`, `get_environment()`, `get_image()`, `get_jobs()`, `get_vanity_url()`, `git`, `permissions`, `set_image_path()`, `set_run_as()`, `set_vanity_url()`, `swap_vanity_url()`, `verify_content_name()`

Other content functions: `acl_add_user()`, `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url_chr()`, `dashboard_url()`, `delete_vanity_url()`, `deploy_repo()`, `get_acl_user()`, `get_environment()`, `get_image()`, `get_jobs()`, `get_vanity_url()`, `git`, `permissions`, `set_image_path()`, `set_run_as()`, `set_vanity_url()`, `swap_vanity_url()`, `verify_content_name()`

`get_content` *Get information about content on the Posit Connect server*

Description

Get information about content on the Posit Connect server

Usage

```
get_content(src, guid = NULL, owner_guid = NULL, name = NULL, ..., .p = NULL)
```

Arguments

`src`	A Connect object
`guid`	The guid for a particular content item
`owner_guid`	The unique identifier of the user who owns the content
`name`	The content name specified when the content was created
`...`	Extra arguments. Currently not used
`.p`	Optional. A predicate function, passed as-is to `purrr::keep()` before turning the response into a tibble. Can be useful for performance

Details

Please see https://docs.posit.co/connect/api/#get-/v1/content for more information

Value

A tibble with the following columns:

- **guid**The unique identifier of this content item.
- **name**A simple, URL-friendly identifier. Allows alpha-numeric characters, hyphens ("-"), and underscores ("_").
- **title**The title of this content.
- **description**A rich description of this content

- **access_type**Access type describes how this content manages its viewers. The value all is the most permissive; any visitor to Posit Connect will be able to view this content. The value logged_in indicates that all Posit Connect accounts may view the content. The acl value lets specifically enumerated users and groups view the content. Users configured as collaborators may always view content. It may have a value of all, logged_in or acl.
- **connection_timeout**Maximum number of seconds allowed without data sent or received across a client connection. A value of 0 means connections will never time-out (not recommended). When null, the default Scheduler.ConnectionTimeout is used. Applies only to content types that are executed on demand.
- **read_timeout**Maximum number of seconds allowed without data received from a client connection. A value of 0 means a lack of client (browser) interaction never causes the connection to close. When null, the default Scheduler.ReadTimeout is used. Applies only to content types that are executed on demand.
- **init_timeout**The maximum number of seconds allowed for an interactive application to start. Posit Connect must be able to connect to a newly launched Shiny application, for example, before this threshold has elapsed. When null, the default Scheduler.InitTimeout is used. Applies only to content types that are executed on demand.
- **idle_timeout**The maximum number of seconds a worker process for an interactive application to remain alive after it goes idle (no active connections). When null, the default Scheduler.IdleTimeout is used. Applies only to content types that are executed on demand.
- **max_processes**Specifies the total number of concurrent processes allowed for a single interactive application. When null, the default Scheduler.MaxProcesses is used. Applies only to content types that are executed on demand.
- **min_processes**Specifies the minimum number of concurrent processes allowed for a single interactive application. When null, the default Scheduler.MinProcesses is used. Applies only to content types that are executed on demand.
- **max_conns_per_process**Specifies the maximum number of client connections allowed to an individual process. Incoming connections which will exceed this limit are routed to a new process or rejected. When null, the default Scheduler.MaxConnsPerProcess is used. Applies only to content types that are executed on demand.
- **load_factor**Controls how aggressively new processes are spawned. When null, the default Scheduler.LoadFactor is used. Applies only to content types that are executed on demand.
- **created_time**The timestamp (RFC3339) indicating when this content was created.
- **last_deployed_time**The timestamp (RFC3339) indicating when this content last had a successful bundle deployment performed.
- **bundle_id**The identifier for the active deployment bundle. Automatically assigned upon the successful deployment of that bundle.
- **app_mode**The runtime model for this content. Has a value of unknown before data is deployed to this item. Automatically assigned upon the first successful bundle deployment. Allowed: api, jupyter-static, python-api, python-bokeh, python-dash, python-streamlit, rmd-shiny, rmd-static, shiny, static, tensorflow-saved-model, unknown
- **content_category**Describes the specialization of the content runtime model. Automatically assigned upon the first successful bundle deployment.
- **parameterized**True when R Markdown rendered content allows parameter configuration. Automatically assigned upon the first successful bundle deployment. Applies only to content with an app_mode of rmd-static.
- **r_version**The version of the R interpreter associated with this content. The value null represents that an R interpreter is not used by this content or that the R package environment has

not been successfully restored. Automatically assigned upon the successful deployment of a bundle.

- **py_version**The version of the Python interpreter associated with this content. The value null represents that a Python interpreter is not used by this content or that the Python package environment has not been successfully restored. Automatically assigned upon the successful deployment of a bundle.
- **run_as**The UNIX user that executes this content. When null, the default Applications.RunAs is used. Applies only to executable content types - not static.
- **run_as_current_user**Indicates if this content is allowed to execute as the logged-in user when using PAM authentication. Applies only to executable content types - not static.
- **owner_guid**The unique identifier for the owner
- **content_url**The URL associated with this content. Computed from the associated vanity URL or GUID for this content.
- **dashboard_url**The URL within the Connect dashboard where this content can be configured. Computed from the GUID for this content.
- **role**The relationship of the accessing user to this content. A value of owner is returned for the content owner. editor indicates a collaborator. The viewer value is given to users who are permitted to view the content. A none role is returned for administrators who cannot view the content but are permitted to view its configuration. Computed at the time of the request.
- **id**The internal numeric identifier of this content item

Examples

```
## Not run:
library(connectapi)
client <- connect()

get_content(client)

## End(Not run)
```

get_content_old *Get information about content on the Posit Connect server*

Description

Get information about content on the Posit Connect server

Usage

```
get_content_old(src, filter = NULL, limit = 25, page_size = 25)
```

Arguments

`src`	The source object
`filter`	a named list of filter options, e.g. list(name = ’appname’)
`limit`	the maximum number of records to return
`page_size`	the number of records to return per page

Details

Please see https://docs.posit.co/connect/api/#getContent for more information

Value

A tibble with the following columns:

- **id**The application ID
- **guid**The unique identifier of this content item.
- **access_type**Access type describes how this content manages its viewers. The value all is the most permissive; any visitor to Posit Connect will be able to view this content. The value logged_in indicates that all Posit Connect accounts may view the content. The acl value lets specifically enumerated users and groups view the content. Users configured as collaborators may always view content. It may have a value of all, logged_in or acl.
- **connection_timeout**Maximum number of seconds allowed without data sent or received across a client connection. A value of 0 means connections will never time-out (not recommended). When null, the default Scheduler.ConnectionTimeout is used. Applies only to content types that are executed on demand.
- **read_timeout**Maximum number of seconds allowed without data received from a client connection. A value of 0 means a lack of client (browser) interaction never causes the connection to close. When null, the default Scheduler.ReadTimeout is used. Applies only to content types that are executed on demand.
- **init_timeout**The maximum number of seconds allowed for an interactive application to start. Posit Connect must be able to connect to a newly launched Shiny application, for example, before this threshold has elapsed. When null, the default Scheduler.InitTimeout is used. Applies only to content types that are executed on demand.
- **idle_timeout**The maximum number of seconds a worker process for an interactive application to remain alive after it goes idle (no active connections). When null, the default Scheduler.IdleTimeout is used. Applies only to content types that are executed on demand.
- **max_processes**Specifies the total number of concurrent processes allowed for a single interactive application. When null, the default Scheduler.MaxProcesses is used. Applies only to content types that are executed on demand.
- **min_processes**Specifies the minimum number of concurrent processes allowed for a single interactive application. When null, the default Scheduler.MinProcesses is used. Applies only to content types that are executed on demand.
- **max_conns_per_process**Specifies the maximum number of client connections allowed to an individual process. Incoming connections which will exceed this limit are routed to a new process or rejected. When null, the default Scheduler.MaxConnsPerProcess is used. Applies only to content types that are executed on demand.
- **load_factor**Controls how aggressively new processes are spawned. When null, the default Scheduler.LoadFactor is used. Applies only to content types that are executed on demand.
- **url**The URL associated with this content. Computed from the associated vanity URL or the identifiers for this content.
- **vanity_url**The vanity url assigned to this app by an administrator
- **name**A simple, URL-friendly identifier. Allows alpha-numeric characters, hyphens ("-"), and underscores ("_").
- **title**The title of this content.

- **bundle_id**The identifier for the active deployment bundle. Automatically assigned upon the successful deployment of that bundle.
- **app_mode**The runtime model for this content. Has a value of unknown before data is deployed to this item. Automatically assigned upon the first successful bundle deployment.
- **content_category**Describes the specialization of the content runtime model. Automatically assigned upon the first successful bundle deployment.
- **has_parameters**True when R Markdown rendered content allows parameter configuration. Automatically assigned upon the first successful bundle deployment. Applies only to content with an app_mode of rmd-static.
- **created_time**The timestamp (RFC3339) indicating when this content was created.
- **last_deployed_time**The timestamp (RFC3339) indicating when this content last had a successful bundle deployment performed.
- **r_version**The version of the R interpreter associated with this content. The value null represents that an R interpreter is not used by this content or that the R package environment has not been successfully restored. Automatically assigned upon the successful deployment of a bundle.
- **py_version**The version of the Python interpreter associated with this content. The value null represents that a Python interpreter is not used by this content or that the Python package environment has not been successfully restored. Automatically assigned upon the successful deployment of a bundle.
- **build_status**
- **run_as**The UNIX user that executes this content. When null, the default Applications.RunAs is used. Applies only to executable content types - not static.
- **run_as_current_user**Indicates if this content is allowed to execute as the logged-in user when using PAM authentication. Applies only to executable content types - not static.
- **description**A rich description of this content
- **app_role**The relationship of the accessing user to this content. A value of owner is returned for the content owner. editor indicates a collaborator. The viewer value is given to users who are permitted to view the content. A none role is returned for administrators who cannot view the content but are permitted to view its configuration. Computed at the time of the request.
- **owner_first_name**The first name of the owner of the content.
- **owner_last_name**The last name of the owner of the content.
- **owner_username**The username of the owner of the content.
- **owner_guid**The unique identifier for the owner
- **owner_email**The email of the content owner
- **owner_locked**Is the owners user account locked?
- **is_scheduled**Is this content scheduled?
- **git**Is this content deployed via git or GitHub?

Examples

```
## Not run:
library(connectapi)
client <- connect()

get_content_old(client, limit = 20)

## End(Not run)
```

`get_environment` *Manage Environment Variables*

Description

Manage Environment Variables for a piece of content.

Usage

```
get_environment(content)

set_environment_new(env, ...)

set_environment_remove(env, ...)

set_environment_all(env, ...)
```

Arguments

`content`	An R6 Content object as returned by `content_item()`
`env`	An R6 Environment object as returned by `get_environment()`
`...`	name = value pairs of environment variable names and values

Details

`get_environment()` returns an Environment object for use with "setter" methods

`set_environment_new()` updates environment values (either creating new values or updating existing). Set `NA` as the value to remove a variable.

`set_environment_remove()` is a wrapper on `set_environment_new()` that allows removing named / listed variables quickly

`set_environment_all()` sets *all* environment variable values (will remove variables not specified)

See Also

Other content functions: `acl_add_user()`, `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url_chr()`, `dashboard_url()`, `delete_vanity_url()`, `deploy_repo()`, `get_acl_user()`, `get_bundles()`, `get_image()`, `get_jobs()`, `get_vanity_url()`, `git`, `permissions`, `set_image_path()`, `set_run_as()`, `set_vanity_url()`, `swap_vanity_url()`, `verify_content_name()`

get_groups *Get group information from the Posit Connect server*

Description

Get group information from the Posit Connect server

Usage

```
get_groups(src, page_size = 20, prefix = NULL, limit = 25)
```

Arguments

`src`	The source object
`page_size`	the number of records to return per page (max 500)
`prefix`	Filters groups by prefix (group name). The filter is case insensitive.
`limit`	The max number of groups to return

Details

Please see https://docs.posit.co/connect/api/#getGroups for more information

Value

A tibble with the following columns:

- **guid**The unique identifier of the group
- **name**The group name
- **owner_guid**The group owner's unique identifier. When using LDAP, or Proxied authentication with group provisioning enabled this property will always be null.

Examples

```
## Not run:
library(connectapi)
client <- connect()

# get all groups
get_groups(client, limit = Inf)

## End(Not run)
```

`get_group_members` *Get users within a specific group*

Description

Get users within a specific group

Usage

```
get_group_members(src, guid)
```

Arguments

`src`	The source object
`guid`	A group GUID identifier

Details

Please see https://docs.posit.co/connect/api/#getGroupMembers for more information

Value

A tibble with the following columns:

- **email**The user's email
- **username**The user's username
- **first_name**The user's first name
- **last_name**The user's last name
- **user_role**The user's role. It may have a value of administrator, publisher or viewer.
- **created_time**The timestamp (in RFC3339 format) when the user was created in the Posit Connect server
- **updated_time**The timestamp (in RFC3339 format) when the user was last updated in the Posit Connect server
- **active_time**The timestamp (in RFC3339 format) when the user was last active on the Posit Connect server
- **confirmed**When false, the created user must confirm their account through an email. This feature is unique to password authentication.
- **locked**Whether or not the user is locked
- **guid**The user's GUID, or unique identifier, in UUID RFC4122 format

Examples

```
## Not run:
library(connectapi)
client <- connect()

# get the first 20 groups
groups <- get_groups(client)
```

```
group_guid <- groups$guid[1]

get_group_members(client, guid = group_guid)

## End(Not run)
```

get_image *Get the Content Image*

Description

[Experimental] `get_image` saves the content image to the given path (default: temp file). `delete_image` removes the image (optionally saving to the given path) `has_image` returns whether the content has an image

Usage

```
get_image(content, path = NULL)

delete_image(content, path = NULL)

has_image(content)
```

Arguments

`content`	A content object
`path`	optional. The path to the image on disk

See Also

Other content functions: `acl_add_user()`, `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url_chr()`, `dashboard_url()`, `delete_vanity_url()`, `deploy_repo()`, `get_acl_user()`, `get_bundles()`, `get_environment()`, `get_jobs()`, `get_vanity_url()`, `git`, `permissions`, `set_image_path()`, `set_run_as()`, `set_vanity_url()`, `swap_vanity_url()`, `verify_content_name()`

get_jobs *Get Jobs*

Description

[Experimental] Retrieve details about jobs associated with a `content_item`. "Jobs" in Posit Connect are content executions

Usage

```
get_jobs(content)

get_job(content, key)
```

Arguments

`content`	A Content object, as returned by `content_item()`
`key`	The key for a job

See Also

Other content functions: `acl_add_user()`, `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url_chr()`, `dashboard_url()`, `delete_vanity_url()`, `deploy_repo()`, `get_acl_user()`, `get_bundles()`, `get_environment()`, `get_image()`, `get_vanity_url()`, `git`, `permissions`, `set_image_path()`, `set_run_as()`, `set_vanity_url()`, `swap_vanity_url()`, `verify_content_name()`

`get_procs` *Get Real-Time Process Data*

Description

[Experimental] This returns real-time process data from the Posit Connect API. It requires administrator privileges to use. NOTE that this only returns data for the server that responds to the request (i.e. in a Highly Available cluster)

Usage

```
get_procs(src)
```

Arguments

`src`	The source object

Value

A tibble with the following columns:

- **pid**The PID of the current process
- **appId**The application ID
- **appGuid**The application GUID
- **appName**The application name
- **appUrl**The application URL
- **appRunAs**The application RunAs user
- **type**The type of process
- **cpuCurrent**The current CPU usage
- **cpuTotal**The total CPU usage
- **ram**The current RAM usage

get_tags — *Get all Tags on the server*

Description

Tag manipulation and assignment functions

Usage

```
get_tags(src)

get_tag_data(src)

create_tag(src, name, parent = NULL)

create_tag_tree(src, ...)

delete_tag(src, tag)

get_content_tags(content)

set_content_tag_tree(content, ...)

set_content_tags(content, ...)

filter_tag_tree_id(tags, ids)

filter_tag_tree_chr(tags, pattern)
```

Arguments

`src`	The source object
`name`	The name of the tag to create
`parent`	optional. A `connect_tag_tree` object (as returned by `get_tags()`) pointed at the parent tag
`...`	Additional arguments

Manage tags (requires Administrator role):

- `get_tags()` - returns a "tag tree" object that can be traversed with `tag_tree$tag1$childtag`
- `get_tag_data()` - returns a tibble of tag data
- `create_tag()` - create a tag by specifying the Parent directly
- `create_tag_tree()` - create tag(s) by specifying the "desired" tag tree hierarchy
- `delete_tag()` - delete a tag (and its children). WARNING: will disassociate any content automatically

Manage content tags:

- `get_content_tags()` - return a `connect_tag_tree` object corresponding to the tags for a piece of content.

- `set_content_tag_tree()` - attach a tag to content by specifying the desired tag tree
- `set_content_tags()` - Set multiple tags at once by providing `connect_tag_tree` objects

Search a tag tree:

- `filter_tag_tree_chr()` - filters a tag tree based on a regex
- `filter_tag_tree_id()` - filters a tag tree based on an id
-

`tag`	A `connect_tag_tree` object (as returned by `get_tags()`)
`content`	An R6 Content object, as returned by `content_item()`
`tags`	A `connect_tag_tree` object (as returned by `get_tags()`)
`ids`	A list of `ids` to filter the tag tree by
`pattern`	A regex to filter the tag tree by (it is passed to `grepl`)

`get_timezones` *Get TimeZones*

Description

Get the available timezones from the server.

Usage

```
get_timezones(connect)
```

Arguments

`connect`	An R6 Connect object

Value

A TimeZone vector to be used for setting time zones

See Also

Other schedule functions: `get_variant_schedule()`, `set_schedule()`

`get_usage_shiny` *Get usage information for deployed shiny applications*

Description

Get usage information for deployed shiny applications

Usage

```
get_usage_shiny(
  src,
  content_guid = NULL,
  min_data_version = NULL,
  from = NULL,
  to = NULL,
  limit = 20,
  previous = NULL,
  nxt = NULL,
  asc_order = TRUE
)
```

Arguments

`src`	the source object
`content_guid`	Filter results by content GUID
`min_data_version`	Filter by data version. Records with a data version lower than the given value will be excluded from the set of results.
`from`	The timestamp that starts the time window of interest. Any usage information that ends prior to this timestamp will not be returned. Individual records may contain a starting time that is before this if they end after it or have not finished. Must be of class Date or POSIX
`to`	The timestamp that ends the time window of interest. Any usage information that starts after this timestamp will not be returned. Individual records may contain an ending time that is after this (or no ending time) if they start before it. Must be of class Date or POSIX
`limit`	The number of records to return.
`previous`	Retrieve the previous page of Shiny application usage logs relative to the provided value. This value corresponds to an internal reference within the server and should be sourced from the appropriate attribute within the paging object of a previous response.
`nxt`	Retrieve the next page of Shiny application usage logs relative to the provided value. This value corresponds to an internal reference within the server and should be sourced from the appropriate attribute within the paging object of a previous response.
`asc_order`	Defaults to TRUE; Determines if the response records should be listed in ascending or descending order within the response. Ordering is by the started timestamp field.

Details

Please see https://docs.posit.co/connect/api/#getShinyAppUsage for more information

Value

A tibble with the following columns:

- **content_guid**The GUID, in RFC4122 format, of the Shiny application this information pertains to.
- **user_guid**The GUID, in RFC4122 format, of the user that visited the application.
- **started**The timestamp, in RFC3339 format, when the user opened the application.
- **ended**The timestamp, in RFC3339 format, when the user left the application.
- **data_version**The data version the record was recorded with. The Shiny Application Events section of the Posit Connect Admin Guide explains how to interpret data_version values.

Examples

```
## Not run:
library(connectapi)
client <- connect()

from <- Sys.Date() - lubridate::days(5)
get_usage_shiny(client, limit = 20, from = from)

## End(Not run)
```

`get_usage_static` *Get usage information from deployed static content*

Description

This function retrieves usage information from static content on the Posit Connect server (e.g. Rmarkdown, Jupyter Notebooks)

Usage

```
get_usage_static(
  src,
  content_guid = NULL,
  min_data_version = NULL,
  from = NULL,
  to = NULL,
  limit = 20,
  previous = NULL,
  nxt = NULL,
  asc_order = TRUE
)
```

Arguments

`src`	the source object
`content_guid`	Filter results by content GUID
`min_data_version`	Filter by data version. Records with a data version lower than the given value will be excluded from the set of results.
`from`	The timestamp that starts the time window of interest. Any usage information that ends prior to this timestamp will not be returned. Individual records may contain a starting time that is before this if they end after it or have not finished. Must be of class Date or POSIX
`to`	The timestamp that ends the time window of interest. Any usage information that starts after this timestamp will not be returned. Individual records may contain an ending time that is after this (or no ending time) if they start before it. Must be of class Date or POSIX
`limit`	The number of records to return.
`previous`	Retrieve the previous page of Shiny application usage logs relative to the provided value. This value corresponds to an internal reference within the server and should be sourced from the appropriate attribute within the paging object of a previous response.
`nxt`	Retrieve the next page of Shiny application usage logs relative to the provided value. This value corresponds to an internal reference within the server and should be sourced from the appropriate attribute within the paging object of a previous response.
`asc_order`	Defaults to TRUE; Determines if the response records should be listed in ascending or descending order within the response. Ordering is by the started timestamp field.

Details

Please see https://docs.posit.co/connect/api/#getContentVisits for more information

Value

A tibble with the following columns:

- **content_guid**The GUID, in RFC4122 format, of the Shiny application this information pertains to.
- **user_guid**The GUID, in RFC4122 format, of the user that visited the application.
- **variant_key**The key of the variant the user visited. This will be null for static content.
- **time**The timestamp, in RFC3339 format, when the user visited the content.
- **rendering_id**The ID of the rendering the user visited. This will be null for static content.
- **bundle_id**The ID of the particular bundle used.
- **data_version**The data version the record was recorded with. The Rendered and Static Content Visit Events section of the Posit Connect Admin Guide explains how to interpret data_version values.

Examples

```
## Not run:
library(connectapi)
client <- connect()

from <- Sys.Date() - lubridate::days(5)
get_usage_static(client, limit = 20, from = from)

## End(Not run)
```

get_users *Get user information from the Posit Connect server*

Description

Get user information from the Posit Connect server

Usage

```
get_users(src, page_size = 20, prefix = NULL, limit = 25)
```

Arguments

`src`	The source object
`page_size`	the number of records to return per page (max 500)
`prefix`	Filters users by prefix (username, first name, or last name). The filter is case insensitive.
`limit`	The max number of records to return

Details

Please see https://docs.posit.co/connect/api/#getUsers for more information

Value

A tibble with the following columns:

- **email**The user's email
- **username**The user's username
- **first_name**The user's first name
- **last_name**The user's last name
- **user_role**The user's role. It may have a value of administrator, publisher or viewer.
- **created_time**The timestamp (in RFC3339 format) when the user was created in the Posit Connect server
- **updated_time**The timestamp (in RFC3339 format) when the user was last updated in the Posit Connect server
- **active_time**The timestamp (in RFC3339 format) when the user was last active on the Posit Connect server

- **confirmed**When false, the created user must confirm their account through an email. This feature is unique to password authentication.
- **locked**Whether or not the user is locked
- **guid**The user's GUID, or unique identifier, in UUID RFC4122 format

Examples

```
## Not run:
library(connectapi)
client <- connect()

# get all users
get_users(client, limit = Inf)

## End(Not run)
```

get_vanity_url *Get the Vanity URL*

Description

Gets the Vanity URL for a piece of content.

Usage

```
get_vanity_url(content)
```

Arguments

content	A Content object

Value

A character string (or NULL if not defined)

See Also

Other content functions: acl_add_user(), content_delete(), content_item(), content_title(), content_update(), create_random_name(), dashboard_url_chr(), dashboard_url(), delete_vanity_url(), deploy_repo(), get_acl_user(), get_bundles(), get_environment(), get_image(), get_jobs(), git, permissions, set_image_path(), set_run_as(), set_vanity_url(), swap_vanity_url(), verify_content_name()

`get_variants` *Get Variant*

Description

[Experimental] Work with variants

Usage

```
get_variants(content)

get_variant_default(content)

get_variant(content, key)
```

Arguments

`content`	An R6 Content object. Returned from `content_item()`
`key`	The Variant key for a specific variant

Details

- `get_variants()` returns a `tibble` with variant data for a `content_item`
- `get_default_variant()` returns the default variant for a `content_item`
- `get_variant()` returns a specific variant for a `content_item` (specified by `key`)

See Also

Other variant functions: `get_variant_renderings()`

Other variant functions: `get_variant_renderings()`

Other variant functions: `get_variant_renderings()`

`get_variant_renderings` *Render a Variant*

Description

[Experimental] Get details about renderings (i.e. render history) or execute a variant on demand

Usage

```
get_variant_renderings(variant)

variant_render(variant)
```

Arguments

`variant`	An R6 Variant object. As returned by `get_variant()` or `get_variant_default()`

Details

- `get_variant_renderings()` returns all renderings / content for a particular variant. Returns a `tibble`
- `variant_render()` executes a variant on demand. Returns a `VariantTask` object

See Also

Other variant functions: `get_variants()`

`get_variant_schedule` *Get a Variant Schedule*

Description

[Experimental] Gets the schedule associated with a Variant.

Usage

```
get_variant_schedule(variant)
```

Arguments

`variant`	A Variant object, as returned by `get_variant()` or `get_variant_default()`

Value

A VariantSchedule object

See Also

Other schedule functions: `get_timezones()`, `set_schedule()`

`git` *Git Repository Helpers*

Description

[Experimental] These functions help use Posit Connect's configured authorization to query available branches and subdirectories for deployment using `deploy_repo()`

Usage

```
repo_check_account(client, host)

repo_check_branches(client, repository)

repo_check_branches_ref(client, repository)

repo_check_manifest_dirs(client, repository, branch)
```

Arguments

`client`	A Connect R6 object
`host`	The git repository host (with schema). For example, "https://github.com"
`repository`	The git repository to explore or consider deploying
`branch`	The git branch to explore for subdirectories

Details

- `repo_check_account()` messages whether an account is in use, and then returns that account
- `repo_check_branches()` retrieves which branches are available, returning in a named list
- `repo_check_manifest_dirs()` retrieves which directories contain a `manifest.json`, returning in a named list

See Also

connectapi::deploy_repo

Other content functions: `acl_add_user()`, `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url_chr()`, `dashboard_url()`, `delete_vanity_url()`, `deploy_repo()`, `get_acl_user()`, `get_bundles()`, `get_environment()`, `get_image()`, `get_jobs()`, `get_vanity_url()`, `permissions`, `set_image_path()`, `set_run_as()`, `set_vanity_url()`, `swap_vanity_url()`, `verify_content_name()`

`groups_create_remote` *Create a Remote Group*

Description

Create a Remote Group

Usage

```
groups_create_remote(connect, prefix, expect = 1, check = TRUE)
```

Arguments

`connect`	A R6 Connect object
`prefix`	character. The prefix of the group name to search for
`expect`	number. The number of responses to expect for this search
`check`	boolean. Whether to check for local existence first

Value

The results of creating the groups

`page_cursor` *Paging*

Description

Helper functions that make paging easier in the Posit Connect Server API.

Helper functions that make paging easier in the Posit Connect Server API.

Usage

```
page_cursor(client, req, limit = Inf)

page_offset(client, req, limit = Inf)
```

Arguments

`client`	A Connect client object
`req`	The request that needs to be paged
`limit`	A row limit

Value

The aggregated results from all requests

The aggregated results from all requests

`permissions` *Content permissions*

Description

Get or set content permissions for a content item

Usage

```
content_add_user(content, guid, role = c("viewer", "owner"))

content_add_group(content, guid, role = c("viewer", "owner"))

content_delete_user(content, guid)

content_delete_group(content, guid)

get_user_permission(content, guid, add_owner = TRUE)

get_my_permission(content, add_owner = TRUE)

get_group_permission(content, guid)

get_content_permissions(content, add_owner = TRUE)
```

Arguments

`content`	An R6 content object
`guid`	The guid associated with either a user (for `content_add_user`) or group (for `content_add_group`)
`role`	The role to assign to a user. Either "viewer" or "owner." Defaults to "viewer"
`add_owner`	Optional. Whether to include the owner in returned permission sets. Default is TRUE. The owner will have an NA_character_ permission "id"

Details

Permission modification:

- `content_add_*` adds a permission to the content
- `content_delete_*` removes a permission from the content

Permission retrieval:

- `get_content_permissions()` lists permissions
- `get_my_permission()` gets the permission associated with the caller.
- `get_user_permission()` gets the permissions associated with a given user. It does not evaluate group memberships
- `get_group_permission()` gets the permissions associated with a given group.

NOTE: by default, the owner is injected with an "NA_character_" permission id. This makes it easier to find / isolate this record.

See Also

Other content functions: `acl_add_user()`, `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url_chr()`, `dashboard_url()`, `delete_vanity_url()`, `deploy_repo()`, `get_acl_user()`, `get_bundles()`, `get_environment()`, `get_image()`, `get_jobs()`, `get_vanity_url()`, `git`, `set_image_path()`, `set_run_as()`, `set_vanity_url()`, `swap_vanity_url()`, `verify_content_name()`

`poll_task` *Poll Task*

Description

Polls a task, waiting for information about a deployment. If the task has results, the output will be a modified "Task" object with `task$get_data()` available to retrieve the results.

Usage

```
poll_task(task, wait = 1, callback = message)
```

Arguments

`task`	A Task object
`wait`	The interval to wait between polling
`callback`	A function to be called for each message received. Set to NULL for no callback

Details

For a simple way to silence messages, set callback = NULL

Value

Task The Task object that was input

See Also

Other deployment functions: bundle_dir(), bundle_path(), bundle_static(), deploy(), download_bundle()

PositConnect *Class representing a Connect API client*

Description

Class representing a Connect API client

Class representing a Connect API client

Usage

```
client <- Connect$new(server = 'connect.example.com',
  apiKey = 'mysecretkey')
client$get_apps()
client$get_tags()
```

Details

This class allows a user to interact with a Connect server via the Connect API. Authentication is done by providing an API key.

Methods

Public methods:

- Connect$get_connect()
- Connect$new()
- Connect$httr_config()
- Connect$print()
- Connect$raise_error()
- Connect$add_auth()
- Connect$GET()
- Connect$GET_RESULT()
- Connect$GET_URL()
- Connect$GET_RESULT_URL()
- Connect$PUT()
- Connect$HEAD()
- Connect$DELETE()
- Connect$PATCH()

- `Connect$POST()`
- `Connect$me()`
- `Connect$get_dashboard_url()`
- `Connect$get_tags()`
- `Connect$get_tag_id()`
- `Connect$get_tag_tree()`
- `Connect$get_tag_tree_old()`
- `Connect$tag_create_safe()`
- `Connect$tag_create()`
- `Connect$tag()`
- `Connect$tag_delete()`
- `Connect$get_n_apps()`
- `Connect$get_apps()`
- `Connect$get_schedule()`
- `Connect$content_create()`
- `Connect$download_bundle()`
- `Connect$bundle_delete()`
- `Connect$content_upload()`
- `Connect$content_deploy()`
- `Connect$content()`
- `Connect$task()`
- `Connect$set_content_tag()`
- `Connect$user()`
- `Connect$users()`
- `Connect$users_remote()`
- `Connect$users_create()`
- `Connect$users_create_remote()`
- `Connect$users_lock()`
- `Connect$users_unlock()`
- `Connect$users_update()`
- `Connect$groups()`
- `Connect$group_members()`
- `Connect$group_member_add()`
- `Connect$group_member_remove()`
- `Connect$groups_create()`
- `Connect$groups_create_remote()`
- `Connect$groups_remote()`
- `Connect$inst_content_visits()`
- `Connect$inst_shiny_usage()`
- `Connect$procs()`
- `Connect$repo_account()`
- `Connect$repo_branches()`
- `Connect$repo_manifest_dirs()`
- `Connect$schedules()`
- `Connect$docs()`

- `Connect$audit_logs()`
- `Connect$server_settings_r()`
- `Connect$server_settings()`
- `Connect$clone()`

Method `get_connect()`**:**

Usage:

```
Connect$get_connect()
```

Method `new()`**:**

Usage:

```
Connect$new(server, api_key)
```

Method `httr_config()`**:**

Usage:

```
Connect$httr_config(...)
```

Method `print()`**:**

Usage:

```
Connect$print(...)
```

Method `raise_error()`**:**

Usage:

```
Connect$raise_error(res)
```

Method `add_auth()`**:**

Usage:

```
Connect$add_auth()
```

Method `GET()`**:**

Usage:

```
Connect$GET(path, writer = httr::write_memory(), parser = "parsed", ...)
```

Method `GET_RESULT()`**:**

Usage:

```
Connect$GET_RESULT(path, writer = httr::write_memory(), ...)
```

Method `GET_URL()`**:**

Usage:

```
Connect$GET_URL(url, writer = httr::write_memory(), parser = "parsed", ...)
```

Method `GET_RESULT_URL()`**:**

Usage:

```
Connect$GET_RESULT_URL(url, writer = httr::write_memory(), ...)
```

Method `PUT()`**:**

Usage:

```
Connect$PUT(path, body, encode = "json", ..., .empty_object = TRUE)
```

Method `HEAD()`**:**

Usage:

```
Connect$HEAD(path, ...)
```

Method `DELETE()`**:**

Usage:

```
Connect$DELETE(path, ...)
```

Method `PATCH()`**:**

Usage:

```
Connect$PATCH(
  path,
  body,
  encode = "json",
  prefix = "/__api__/",
  ...,
  .empty_object = TRUE
)
```

Method `POST()`**:**

Usage:

```
Connect$POST(
  path,
  body,
  encode = "json",
  prefix = "/__api__/",
  ...,
  .empty_object = TRUE
)
```

Method `me()`**:**

Usage:

```
Connect$me()
```

Method `get_dashboard_url()`**:**

Usage:

```
Connect$get_dashboard_url()
```

Method `get_tags()`**:**

Usage:

```
Connect$get_tags(use_cache = FALSE)
```

Method `get_tag_id()`**:**

Usage:

```
Connect$get_tag_id(tagname)
```

Method `get_tag_tree()`**:**

Usage:

```
Connect$get_tag_tree()
```

Method `get_tag_tree_old()`**:**

Usage:

```
Connect$get_tag_tree_old()
```

Method `tag_create_safe()`**:**

Usage:

```
Connect$tag_create_safe(name, parent_id = NULL)
```

Method `tag_create()`**:**

Usage:

```
Connect$tag_create(name, parent_id = NULL)
```

Method `tag()`**:**

Usage:

```
Connect$tag(id = NULL)
```

Method `tag_delete()`**:**

Usage:

```
Connect$tag_delete(id)
```

Method `get_n_apps()`**:**

Usage:

```
Connect$get_n_apps()
```

Method `get_apps()`**:**

Usage:

```
Connect$get_apps(filter = NULL, .collapse = "&", .limit = Inf, page_size = 25)
```

Method `get_schedule()`**:**

Usage:

```
Connect$get_schedule(schedule_id)
```

Method `content_create()`**:**

Usage:

```
Connect$content_create(name, title = name, ...)
```

Method `download_bundle()`**:**

Usage:

```
Connect$download_bundle(bundle_id, to_path = tempfile(), overwrite = FALSE)
```

Method `bundle_delete()`**:**

Usage:

```
Connect$bundle_delete(bundle_id)
```

Method `content_upload()`**:**

Usage:

```
Connect$content_upload(bundle_path, guid)
```

Method `content_deploy()`**:**

Usage:

```
Connect$content_deploy(guid, bundle_id)
```

Method `content()`:

Usage:

```
Connect$content(
  guid = NULL,
  owner_guid = NULL,
  name = NULL,
  include = "tags,owner"
)
```

Method `task()`:

Usage:

```
Connect$task(task_id, first = 0, wait = 5)
```

Method `set_content_tag()`:

Usage:

```
Connect$set_content_tag(content_id, tag_id)
```

Method `user()`:

Usage:

```
Connect$user(guid)
```

Method `users()`:

Usage:

```
Connect$users(page_number = 1, prefix = NULL, page_size = 20)
```

Method `users_remote()`:

Usage:

```
Connect$users_remote(prefix)
```

Method `users_create()`:

Usage:

```
Connect$users_create(
  username,
  email,
  first_name = NULL,
  last_name = NULL,
  password = NULL,
  user_must_set_password = NULL,
  user_role = NULL,
  unique_id = NULL
)
```

Method `users_create_remote()`:

Usage:

```
Connect$users_create_remote(temp_ticket)
```

Method `users_lock()`:

Usage:

```
Connect$users_lock(user_guid)
```

Method `users_unlock()`:

Usage:

```
Connect$users_unlock(user_guid)
```

Method `users_update()`:

Usage:

```
Connect$users_update(user_guid, ...)
```

Method `groups()`:

Usage:

```
Connect$groups(page_number = 1, prefix = NULL, page_size = 20)
```

Method `group_members()`:

Usage:

```
Connect$group_members(guid)
```

Method `group_member_add()`:

Usage:

```
Connect$group_member_add(group_guid, user_guid)
```

Method `group_member_remove()`:

Usage:

```
Connect$group_member_remove(group_guid, user_guid)
```

Method `groups_create()`:

Usage:

```
Connect$groups_create(name)
```

Method `groups_create_remote()`:

Usage:

```
Connect$groups_create_remote(temp_ticket)
```

Method `groups_remote()`:

Usage:

```
Connect$groups_remote(prefix = NULL, limit = 20)
```

Method `inst_content_visits()`:

Usage:

```
Connect$inst_content_visits(
  content_guid = NULL,
  min_data_version = NULL,
  from = NULL,
  to = NULL,
  limit = 20,
  previous = NULL,
  nxt = NULL,
  asc_order = TRUE
)
```

Method `inst_shiny_usage()`:

Usage:

```
Connect$inst_shiny_usage(
  content_guid = NULL,
  min_data_version = NULL,
  from = NULL,
  to = NULL,
  limit = 20,
  previous = NULL,
  nxt = NULL,
  asc_order = TRUE
)
```

Method `procs()`:

Usage:

```
Connect$procs()
```

Method `repo_account()`:

Usage:

```
Connect$repo_account(host)
```

Method `repo_branches()`:

Usage:

```
Connect$repo_branches(repo)
```

Method `repo_manifest_dirs()`:

Usage:

```
Connect$repo_manifest_dirs(repo, branch)
```

Method `schedules()`:

Usage:

```
Connect$schedules(
  start = Sys.time(),
  end = Sys.time() + 60 * 60 * 24 * 7,
  detailed = FALSE
)
```

Method `docs()`:

Usage:

```
Connect$docs(docs = "api", browse = TRUE)
```

Method `audit_logs()`:

Usage:

```
Connect$audit_logs(limit = 20L, previous = NULL, nxt = NULL, asc_order = TRUE)
```

Method `server_settings_r()`:

Usage:

```
Connect$server_settings_r()
```

Method `server_settings()`:

Usage:

```
Connect$server_settings()
```

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

Usage:

```
Connect$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other R6 classes: `Bundle`, `ContentTask`, `Content`, `Environment`, `Task`, `Vanity`, `VariantSchedule`, `VariantTask`, `Variant`

`promote` *Promote content from one Connect server to another*

Description

Promote content from one Connect server to another

Usage

```
promote(from, to, to_key, from_key, name)
```

Arguments

`from`	The url for the server containing the content (the originating server)
`to`	The url for the server where the content will be deployed (the destination server)
`to_key`	An API key on the destination "to" server. If the destination content is going to be updated, the API key must belong to a user with collaborator access on the content that will be updated. If the destination content is to be created new, the API key must belong to a user with publisher privileges.
`from_key`	An API key on the originating "from" server. The API key must belong to a user with collaborator access to the content to be promoted.
`name`	The name of the content on the originating "from" server. If content with the same name is found on the destination server, the content will be updated. If no content on the destination server has a matching name, a new endpoint will be created.

Value

The URL for the content on the destination "to" server

`set_image_path` *Set the Content Image*

Description

[Experimental]

Usage

```
set_image_path(content, path)

set_image_url(content, url)

set_image_webshot(content, ...)
```

Arguments

`content`	A content object
`path`	The path to an image on disk
`url`	The url for an image
`...`	Additional arguments passed on to `webshot2::webshot()`

Details

Set the Content Image using a variety of methods.

NOTE: `set_image_webshot()` requires `webshot2::webshot()`, but currently skips and warns for any content that requires authentication until the `webshot2` package supports authentication.

See Also

Other content functions: `acl_add_user()`, `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url_chr()`, `dashboard_url()`, `delete_vanity_url()`, `deploy_repo()`, `get_acl_user()`, `get_bundles()`, `get_environment()`, `get_image()`, `get_jobs()`, `get_vanity_url()`, `git`, `permissions`, `set_run_as()`, `set_vanity_url()`, `swap_vanity_url()`, `verify_content_name()`

`set_run_as` *Set RunAs User*

Description

Set the `RunAs` user for a piece of content. The `run_as_current_user` flag only does anything if:

Usage

```
set_run_as(content, run_as, run_as_current_user = FALSE)
```

Arguments

`content`	an R6 Content item
`run_as`	The RunAs user to use for this content
`run_as_current_user`	Whether to run this content as the viewer of the application

Details

- PAM is the authentication method
- `Applications.RunAsCurrentUser` is enabled on the server

Also worth noting that the `run_as` user must exist on the Posit Connect server (as a linux user) and have appropriate group memberships, or you will get a `400: Bad Request`. Set to `NULL` to use the default RunAs user / unset any current configuration.

To "read" the current RunAs user, use the `Content` object or `get_content()` function.

Value

a Content object, updated with new details

See Also

connectapi::content_update

Other content functions: `acl_add_user()`, `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url_chr()`, `dashboard_url()`, `delete_vanity_url()`, `deploy_repo()`, `get_acl_user()`, `get_bundles()`, `get_environment()`, `get_image()`, `get_jobs()`, `get_vanity_url()`, `git`, `permissions`, `set_image_path()`, `set_vanity_url()`, `swap_vanity_url()`, `verify_content_name()`

`set_schedule` *Set a Schedule*

Description

[Experimental] Sets the schedule for a given Variant. Requires a `Schedule` object (as returned by `get_variant_schedule()`)

Usage

```
set_schedule(.schedule, ...)

set_schedule_minute(
  .schedule,
  n = 30,
  start_time = Sys.time(),
  activate = TRUE,
  email = FALSE,
  timezone = Sys.timezone()
)
```

```
set_schedule_hour(
  .schedule,
  n = 1,
  start_time = Sys.time(),
  activate = TRUE,
  email = FALSE,
  timezone = Sys.timezone()
)

set_schedule_day(
  .schedule,
  n = 1,
  start_time = Sys.time(),
  activate = TRUE,
  email = FALSE,
  timezone = Sys.timezone()
)

set_schedule_weekday(
  .schedule,
  start_time = Sys.time(),
  activate = TRUE,
  email = FALSE,
  timezone = Sys.timezone()
)

set_schedule_week(
  .schedule,
  n = 1,
  start_time = Sys.time(),
  activate = TRUE,
  email = FALSE,
  timezone = Sys.timezone()
)

set_schedule_dayofweek(
  .schedule,
  days,
  start_time = Sys.time(),
  activate = TRUE,
  email = FALSE,
  timezone = Sys.timezone()
)

set_schedule_semimonth(
  .schedule,
  first = TRUE,
  start_time = Sys.time(),
  activate = TRUE,
  email = FALSE,
  timezone = Sys.timezone()
)
```

```
set_schedule_dayofmonth(
  .schedule,
  n = 1,
  day = 1,
  start_time = Sys.time(),
  activate = TRUE,
  email = FALSE,
  timezone = Sys.timezone()
)

set_schedule_dayweekofmonth(
  .schedule,
  n = 1,
  day = 1,
  week = 1,
  start_time = Sys.time(),
  activate = TRUE,
  email = FALSE,
  timezone = Sys.timezone()
)

set_schedule_year(
  .schedule,
  n = 1,
  start_time = Sys.time(),
  activate = TRUE,
  email = FALSE,
  timezone = Sys.timezone()
)

set_schedule_remove(.schedule)

schedule_describe(.schedule)
```

Arguments

`.schedule`	A schedule object. As returned by `get_variant_schedule()`
`...`	Scheduling parameters
`n`	The "number of" iterations
`start_time`	The start time of the schedule
`activate`	Whether to publish the output of this schedule
`email`	Whether to send emails on this schedule
`timezone`	The timezone to use for setting the schedule. Defaults to `Sys.timezone()`
`days`	The days of the week (0-6)
`first`	logical Whether to execute on the 1st and 15th (TRUE) or 14th and last (FALSE)
`day`	The day of the week (0-6) or day of the month (0-31)
`week`	The week of the month (0-5)
`schedule`	A JSON blob (as a string) describing the schedule. See "More Details"

Details

- `set_schedule()` is a raw interface to Posit Connect's `schedule` API
- `set_schedule_*()` functions provide handy wrappers around `set_schedule()`
- `set_schedule_remove()` removes a schedule / un-schedules a variant

Beware, using `set_schedule()` currently uses the Posit Connect `schedule` API directly, and so can be a little clunky. Using the `set_schedule_*()` is generally recommended.

Value

An updated Schedule object

See Also

Other schedule functions: `get_timezones()`, `get_variant_schedule()`

`set_vanity_url` *Set the Vanity URL*

Description

Sets the Vanity URL for a piece of content.

Usage

```
set_vanity_url(content, url, force = FALSE)
```

Arguments

`content`	A Content object
`url`	The path component of the URL
`force`	optional. Default FALSE. Whether to force-reassign a vanity URL that already exists

Value

An updated Content object

See Also

Other content functions: `acl_add_user()`, `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url_chr()`, `dashboard_url()`, `delete_vanity_url()`, `deploy_repo()`, `get_acl_user()`, `get_bundles()`, `get_environment()`, `get_image()`, `get_jobs()`, `get_vanity_url()`, `git`, `permissions`, `set_image_path()`, `set_run_as()`, `swap_vanity_url()`, `verify_content_name()`

Examples

```
## Not run:
bnd <- bundle_dir("~/my/directory")
connect() %>%
  deploy(bnd) %>%
  set_vanity_url("a/vanity/url")

## End(Not run)
```

`swap_vanity_url` *Swap the Vanity URL*

Description

Swaps the Vanity URLs between two pieces of content

Usage

```
swap_vanity_url(from_content, to_content)
```

Arguments

`from_content`	A Content object
`to_content`	A Content object

See Also

Other content functions: `acl_add_user()`, `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url_chr()`, `dashboard_url()`, `delete_vanity_url()`, `deploy_repo()`, `get_acl_user()`, `get_bundles()`, `get_environment()`, `get_image()`, `get_jobs()`, `get_vanity_url()`, `git`, `permissions`, `set_image_path()`, `set_run_as()`, `set_vanity_url()`, `verify_content_name()`

`Task` *Task*

Description

An R6 class that represents a Task

Methods

Public methods:

- `Task$new()`
- `Task$get_connect()`
- `Task$get_task()`
- `Task$add_data()`
- `Task$get_data()`

- `Task$print()`
- `Task$clone()`

Method `new()`**:**

Usage:

```
Task$new(connect, task)
```

Method `get_connect()`**:**

Usage:

```
Task$get_connect()
```

Method `get_task()`**:**

Usage:

```
Task$get_task()
```

Method `add_data()`**:**

Usage:

```
Task$add_data(data)
```

Method `get_data()`**:**

Usage:

```
Task$get_data()
```

Method `print()`**:**

Usage:

```
Task$print(...)
```

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

Usage:

```
Task$clone(deep = FALSE)
```

Arguments:

`deep`	Whether to make a deep clone.

See Also

Other R6 classes: `Bundle`, `ContentTask`, `Content`, `Environment`, `PositConnect`, `Vanity`, `VariantSchedule`, `VariantTask`, `Variant`

tbl_connect *Connect Tibble*

Description

[Experimental] A lazy tibble that automatically pages through API requests when `collected`.

Usage

```
tbl_connect(
  src,
  from = c("users", "groups", "content", "usage_shiny", "usage_static", "audit_logs"),
  ...
)
```

Arguments

`src`	The source object
`from`	The type of tibble
`...`	Additional arguments that are not yet implemented

Value

A `tbl_connect` object

users_create_remote *Create a Remote User*

Description

The remote user creation workflow involves authentication providers like LDAP that involve a queryable identity store. This helper wraps the API calls necessary to retrieve information about and then create such a user. It functions with a "fuzzy match" `prefix` by default, but if you want to instantiate users directly, you should set `exact = TRUE`.

Usage

```
users_create_remote(connect, prefix, expect = 1, check = TRUE, exact = FALSE)
```

Arguments

`connect`	A R6 Connect object
`prefix`	character. The prefix of the user name to search for
`expect`	number. Optional. The number of responses to expect for this search
`check`	boolean. Optional. Whether to check for local existence first
`exact`	boolean. Optional. Whether to only create users whose username exactly matches the provided `prefix`.

Details

NOTE: there can be problems with usernames that are not unique. Please open an issue if you run into any problems.

Value

The results of creating the users

`user_guid_from_username` *User*

Description

Get user details

Usage

```
user_guid_from_username(client, username)
```

Arguments

`client`	A Connect R6 object
`username`	The user's username

Details

`user_guid_from_username()` is a helper to retrieve a user GUID, given the user's username. It is useful in Shiny applications for using `session$user`

`Vanity` *Vanity*

Description

An R6 class that represents a Vanity URL

Super class

`connectapi::Content` -> `Vanity`

Methods

Public methods:

- `Vanity$new()`
- `Vanity$get_vanity()`
- `Vanity$print()`
- `Vanity$clone()`

Method `new()`**:**

Usage:

```
Vanity$new(connect, content, vanity)
```

Method `get_vanity()`**:**

Usage:

```
Vanity$get_vanity()
```

Method `print()`**:**

Usage:

```
Vanity$print(...)
```

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

Usage:

```
Vanity$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other R6 classes: `Bundle`, `ContentTask`, `Content`, `Environment`, `PositConnect`, `Task`, `VariantSchedule`, `VariantTask`, `Variant`

`Variant` *Variant*

Description

An R6 class that represents a Variant

Super class

`connectapi::Content` -> `Variant`

Methods

Public methods:

- `Variant$get_variant()`
- `Variant$get_variant_remote()`
- `Variant$new()`
- `Variant$send_mail()`
- `Variant$get_schedule()`
- `Variant$get_schedule_remote()`
- `Variant$get_subscribers()`
- `Variant$remove_subscriber()`
- `Variant$add_subscribers()`
- `Variant$render()`
- `Variant$renderings()`
- `Variant$update_variant()`
- `Variant$jobs()`
- `Variant$job()`
- `Variant$get_url()`
- `Variant$get_url_rev()`
- `Variant$get_dashboard_url()`
- `Variant$print()`
- `Variant$clone()`

Method `get_variant()`**:**

Usage:

```
Variant$get_variant()
```

Method `get_variant_remote()`**:**

Usage:

```
Variant$get_variant_remote()
```

Method `new()`**:**

Usage:

```
Variant$new(connect, content, key)
```

Method `send_mail()`**:**

Usage:

```
Variant$send_mail(to = c("me", "collaborators", "collaborators_viewers"))
```

Method `get_schedule()`**:**

Usage:

```
Variant$get_schedule()
```

Method `get_schedule_remote()`**:**

Usage:

```
Variant$get_schedule_remote()
```

Method `get_subscribers()`**:**

Usage:

`Variant$get_subscribers()`

Method `remove_subscriber()`**:**

Usage:

`Variant$remove_subscriber(guid)`

Method `add_subscribers()`**:**

Usage:

`Variant$add_subscribers(guids)`

Method `render()`**:**

Usage:

`Variant$render()`

Method `renderings()`**:**

Usage:

`Variant$renderings()`

Method `update_variant()`**:**

Usage:

`Variant$update_variant(...)`

Method `jobs()`**:**

Usage:

`Variant$jobs()`

Method `job()`**:**

Usage:

`Variant$job(key)`

Method `get_url()`**:**

Usage:

`Variant$get_url()`

Method `get_url_rev()`**:**

Usage:

`Variant$get_url_rev(rev)`

Method `get_dashboard_url()`**:**

Usage:

`Variant$get_dashboard_url(pane = "access")`

Method `print()`**:**

Usage:

`Variant$print(...)`

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

Usage:

`Variant$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

See Also

Other R6 classes: `Bundle`, `ContentTask`, `Content`, `Environment`, `PositConnect`, `Task`, `Vanity`, `VariantSchedule`, `VariantTask`

`VariantSchedule` *VariantSchedule*

Description

An R6 class that represents a Schedule

Super classes

`connectapi::Content` -> `connectapi::Variant` -> `VariantSchedule`

Methods

Public methods:

- `VariantSchedule$new()`
- `VariantSchedule$GET()`
- `VariantSchedule$POST()`
- `VariantSchedule$DELETE()`
- `VariantSchedule$set_schedule()`
- `VariantSchedule$is_empty()`
- `VariantSchedule$print()`
- `VariantSchedule$get_schedule()`
- `VariantSchedule$get_schedule_remote()`
- `VariantSchedule$describe_schedule()`
- `VariantSchedule$clone()`

Method `new()`**:**

Usage:

```
VariantSchedule$new(connect, content, key, schedule)
```

Method `GET()`**:**

Usage:

```
VariantSchedule$GET(path)
```

Method `POST()`**:**

Usage:

```
VariantSchedule$POST(path, body)
```

Method `DELETE()`**:**

Usage:

```
VariantSchedule$DELETE(path)
```

Method `set_schedule()`**:**

Usage:

```
VariantSchedule$set_schedule(...)
```

Method `is_empty()`**:**

Usage:

```
VariantSchedule$is_empty()
```

Method `print()`**:**

Usage:

```
VariantSchedule$print(...)
```

Method `get_schedule()`**:**

Usage:

```
VariantSchedule$get_schedule()
```

Method `get_schedule_remote()`**:**

Usage:

```
VariantSchedule$get_schedule_remote()
```

Method `describe_schedule()`**:**

Usage:

```
VariantSchedule$describe_schedule()
```

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

Usage:

```
VariantSchedule$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other R6 classes: `Bundle`, `ContentTask`, `Content`, `Environment`, `PositConnect`, `Task`, `Vanity`, `VariantTask`, `Variant`

`VariantTask` *VariantTask*

Description

An R6 class that represents a Variant Task

Super classes

`connectapi::Content` -> `connectapi::Variant` -> `VariantTask`

Methods

Public methods:

- `VariantTask$new()`
- `VariantTask$get_task()`
- `VariantTask$add_data()`
- `VariantTask$get_data()`
- `VariantTask$print()`
- `VariantTask$clone()`

Method `new()`**:**

Usage:

```
VariantTask$new(connect, content, key, task)
```

Method `get_task()`**:**

Usage:

```
VariantTask$get_task()
```

Method `add_data()`**:**

Usage:

```
VariantTask$add_data(data)
```

Method `get_data()`**:**

Usage:

```
VariantTask$get_data()
```

Method `print()`**:**

Usage:

```
VariantTask$print(...)
```

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

Usage:

```
VariantTask$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other R6 classes: `Bundle`, `ContentTask`, `Content`, `Environment`, `PositConnect`, `Task`, `Vanity`, `VariantSchedule`, `Variant`

`vec_cast.fs_bytes` *Cast to fs_bytes*

Description

[Deprecated] This is a temporary placeholder because the functionality does not exist yet in the `fs` package. Do not build dependencies on `connectapi::vec-cast.fs_bytes`, as it will be removed without warning in a future release.

Usage

```
vec_cast.fs_bytes(x, to, ...)
```

Arguments

`x`	Vectors to cast
`to`	Type to cast to. If `NULL`, `x` will be returned as is
`...`	Dots for future extensions and should be empty

Value

A vector the same length as `x` with the same type as `to`, or an error if the cast is not possible.

`verify_content_name` *Verify Content Name*

Description

Ensures that a content name fits the specifications / requirements of Posit Connect. Throws an error if content name is invalid. Content names (as of the time of writing) must be between 3 and 64 alphanumeric characters, dashes, and underscores

Usage

```
verify_content_name(name)
```

Arguments

`name`	The proposed content name

Value

The name (or an error if invalid)

See Also

connectapi::create_random_name

Other content functions: `acl_add_user()`, `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url_chr()`, `dashboard_url()`, `delete_vanity_url()`, `deploy_repo()`, `get_acl_user()`, `get_bundles()`, `get_environment()`, `get_image()`, `get_jobs()`, `get_vanity_url()`, `git`, `permissions`, `set_image_path()`, `set_run_as()`, `set_vanity_url()`, `swap_vanity_url()`