Package 'connectapi'

February 25, 2026

Type Package

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

Version 0.11.0

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://posit-dev.github.io/connectapi/, https://github.com/posit-dev/connectapi

BugReports https://github.com/posit-dev/connectapi/issues

Imports bit64, fs, glue, httr, mime, jsonlite, lifecycle, magrittr, purrr, R6, rlang (>= 0.4.2), tibble, uuid, vctrs (>= 0.3.0), base64enc

Suggests covr, dbplyr, dplyr, ggplot2, gridExtra, httptest, knitr, lubridate, progress, rmarkdown, rprojroot, rsconnect, spelling, testthat, tidyr, webshot2, withr

VignetteBuilder knitr

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

Config/testthat/edition 3

Collate 'audits.R' 'browse.R' 'connect.R' 'connectapi-package.R' 'connectapi.R' 'content.R' 'deploy.R' 'get.R' 'git.R' 'groups.R' 'integrations.R' 'lazy.R' 'page.R' 'parse.R' 'promote.R' 'ptype.R' 'remote.R' 'runtime-caches.R' 'schedule.R' 'tags.R' 'utils.R' 'thumbnail.R' 'user.R' 'utils-pipe.R' 'variant.R'

NeedsCompilation no

Author Kara Woo [aut, cre],
Toph Allen [aut],
Neal Richardson [aut],
Sean Lopp [aut],
Cole Arendt [aut],
Posit, PBC [cph, fnd]

Maintainer Kara Woo <kara.woo@posit.co>

Repository CRAN

Date/Publication 2026-02-25 20:40:02 UTC

Contents

`as.data.frame.connect_content_list`
Convert content list to a data frame

Description

Converts a list returned by `search_content()` into a data frame.

Usage

```
## S3 method for class 'connect_content_list'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	A `connect_content_list` object (from `search_content()`).
`row.names`	Passed to `base::as.data.frame()`.
`optional`	Passed to `base::as.data.frame()`.
`...`	Passed to `base::as.data.frame()`.

Value

A `data.frame` with one row per content item.

as.data.frame.connect_integration_list

Convert integrations list to a data frame

Description

Converts a list returned by `get_integrations()` into a data frame.

Usage

```
## S3 method for class 'connect_integration_list'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	A `connect_integration_list` object (from `get_integrations()`).
`row.names`	Passed to `base::as.data.frame()`.
`optional`	Passed to `base::as.data.frame()`.
`...`	Passed to `base::as.data.frame()`.

Value

A `data.frame` with one row per integration.

as.data.frame.connect_list_hits

Convert usage data to a data frame

Description

Converts an object returned by `get_usage()` into a data frame with parsed column types. By default, extracts `path` and `user_agent` from the `data` field, if available.

Usage

```
## S3 method for class 'connect_list_hits'
as.data.frame(x, row.names = NULL, optional = FALSE, ..., unnest = TRUE)
```

Arguments

`x`	A `connect_list_hits` object (from `get_usage()`).
`row.names`	Passed to `base::as.data.frame()`.
`optional`	Passed to `base::as.data.frame()`.
`...`	Passed to `base::as.data.frame()`.
`unnest`	Logical; if `TRUE` (default), extracts nested fields using **tidyr**. Set to `FALSE` to skip unnesting.

Value

A `data.frame` with one row per usage record.

`as_integration` *Convert objects to integration class*

Description

Convert objects to integration class

Usage

```
as_integration(x, client)
```

Arguments

`x`	An object to convert to an integration.
`client`	The Connect client object where the integration comes from.

Value

An integration object. The object has all the fields from the integrations endpoint (see `get_integrations()`) and a Connect client as a `client` attribute (`attr(x, "client")`)

`as_tibble.connect_content_list`
Convert integration list to a tibble

Description

Converts a list returned by `search_content()` to a tibble.

Usage

```
## S3 method for class 'connect_content_list'
as_tibble(x, ...)
```

Arguments

`x`	A `connect_content_list` object.
`...`	Unused.

Value

A tibble with one row per content item.

as_tibble.connect_integration_list

Convert integration list to a tibble

Description

Converts a list returned by `get_integrations()` to a tibble.

Usage

```
## S3 method for class 'connect_integration_list'
as_tibble(x, ...)
```

Arguments

`x`	A `connect_integration_list` object.
`...`	Unused.

Value

A tibble with one row per integration.

as_tibble.connect_list_hits

Convert usage data to a tibble

Description

Converts an object returned by `get_usage()` to a tibble via `as.data.frame.connect_list_hits()`.

Usage

```
## S3 method for class 'connect_list_hits'
as_tibble(x, ...)
```

Arguments

`x`	A `connect_list_hits` object.
`...`	Passed to `as.data.frame()`.

Value

A tibble with one row per usage record.

audit_access_open *Audit Access Controls*

Description

[Experimental]

Usage

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

Arguments

content	data.frame of content information, as from get_content()
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(), vanity_is_available()

audit_runas *Audit Run As Settings*

Description

[Experimental]

Usage

```
audit_runas(content)
```

Arguments

content	data.frame of content information, as from get_content()

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(), vanity_is_available()

audit_r_versions *Audit R Versions*

Description

[Experimental]

Usage

```
audit_r_versions(content)
```

Arguments

content	data.frame of content information, as from get_content()

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(), vanity_is_available()

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

Bundle

Bundle

Details

An R6 class that represents a bundle

Public fields

`path` The bundle path on disk.

`size` The size of the bundle.

Methods

Public methods:

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

Method `new()`**:** Initialize this content bundle.

Usage:

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

Arguments:

`path` The bundle path on disk.

Method `print()`**:** Print this object.

Usage:

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

Arguments:

`...` Unused.

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

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()

Examples

```
bundle_dir(system.file("tests/testthat/examples/shiny/", package = "connectapi"))
```

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()

Examples

```
bundle_path(system.file("tests/testthat/examples/static.tar.gz", package = "connectapi"))
```

---

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()`

Examples

```
bundle_static(system.file("logo.png", package = "connectapi"))
```

`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("CONNECT_SERVER"),
  api_key = Sys.getenv("CONNECT_API_KEY"),
  token,
  token_local_testing_key = api_key,
  audience = NULL,
  ...,
  .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
`token`	Optional. A user session token. When running on a Connect server, creates a client using the content visitor's account. Running locally, the created client uses the provided API key.
`token_local_testing_key`	Optional. Only used when not running on Connect and a `token` is provided. By default, the function returns a `Connect` object using the `api_key`. By providing a different key here you can test a visitor client with differently-scoped permissions.
`audience`	Optional. The GUID of a Connect API integration associated with this piece of content.

`...` 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.

Details

When running on Connect, the client's environment will contain default `CONNECT_SERVER` and `CONNECT_API_KEY` variables. The API key's permissions are scoped to the publishing user's account.

To create a client with permissions scoped to the content visitor's account, call `connect()` passing a user session token from content session headers to the `token` argument. To do this, you must first add a Connect API integration in your published content's Access sidebar.

Value

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

Examples

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

# Running in Connect, create a client using the content visitor's account.
# This example assumes code is being executed in a Shiny app's `server`
# function with a `session` object available.
token <- session$request$HTTP_POSIT_CONNECT_USER_SESSION_TOKEN
client <- connect(token = token)

# Test locally with an API key using a different role.
fallback_key <- Sys.getenv("VIEWER_ROLE_API_KEY")
client <- connect(token = token, token_local_testing_key = fallback_key)

## End(Not run)



# default is to read CONNECT_SERVER and CONNECT_API_KEY environment variables
connect()
```

`Content` *Content*

Description

Content

Content

Details

An R6 class that represents content.

Public fields

`connect` An R6 Connect object.

`content` The content details from Posit Connect. Properties are described in `get_content()`.

Active bindings

`default_variant` The default variant for this object.

`is_rendered` TRUE if this is a rendered content type, otherwise FALSE.

`is_interactive` TRUE if this is a rendered content type, otherwise FALSE.

Methods

Public methods:

- `Content$new()`
- `Content$get_content_remote()`
- `Content$get_bundles()`
- `Content$bundle_download()`
- `Content$bundle_delete()`
- `Content$update()`
- `Content$danger_delete()`
- `Content$get_url()`
- `Content$get_dashboard_url()`
- `Content$jobs()`
- `Content$register_job_kill_order()`
- `Content$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$deploy()`
- `Content$repository()`
- `Content$repo_enable()`
- `Content$repo_set()`

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

Method `new()`**:** Initialize this content.

Usage:

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

Arguments:

`connect` The `Connect` instance.

`content` The content data.

Method `get_content_remote()`**:** Obtain the content data from the Connect server.

Usage:

```
Content$get_content_remote()
```

Method `get_bundles()`**:** Return the set of content bundles.

Usage:

```
Content$get_bundles()
```

Method `bundle_download()`**:** Download the source archive for a content bundle.

Usage:

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

Arguments:

`bundle_id` The bundle identifer.

`filename` Where to write the result.

`overwrite` Overwrite an existing filename.

Method `bundle_delete()`**:** Delete a content bundle.

Usage:

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

Arguments:

`bundle_id` The bundle identifer.

Method `update()`**:** Update this content item.

Usage:

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

Arguments:

`...` Content fields.

Method `danger_delete()`**:** Delete this content item.

Usage:

```
Content$danger_delete()
```

Method `get_url()`**:** Return the URL for this content.

Usage:

```
Content$get_url()
```

Method `get_dashboard_url()`**:** Return the URL for this content in the Posit Connect dashboard.

Usage:

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

Arguments:

`pane` The pane in the dashboard to link to.

Method `jobs()`**:** Return the jobs for this content

Usage:

```
Content$jobs()
```

Method `register_job_kill_order()`**:** Terminate a single job for this content item.

Usage:

```
Content$register_job_kill_order(key)
```

Arguments:

`key` The job key.

Method `variants()`**:** Return the variants for this content.

Usage:

```
Content$variants()
```

Method `tag_set()`**:** Set a tag for this content.

Usage:

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

Arguments:

`tag_id` The tag identifier.

Method `tag_delete()`**:** Remove a tag for this content.

Usage:

```
Content$tag_delete(tag_id)
```

Arguments:

`tag_id` The tag identifier.

Method `tags()`**:** The tags for this content.

Usage:

```
Content$tags()
```

Method `permissions_add()`**:** Add a principal to the ACL for this content.

Usage:

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

Arguments:

`principal_guid` GUID for the target user or group.
`principal_type` Acting on user or group.
`role` The kind of content access.

Method `permissions_update()`**:** Alter a principal in the ACL for this content.

Usage:

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

Arguments:

`id` The target identifier.
`principal_guid` GUID for the target user or group.
`principal_type` Acting on user or group.
`role` The kind of content access.

Method `permissions_delete()`**:** Remove an entry from the ACL for this content.

Usage:

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

Arguments:

`id` The target identifier.

Method `permissions()`**:** Obtain some or all of the ACL for this content.

Usage:

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

Arguments:

`id` The target identifier.
`add_owner` Include the content owner in the result set.

Method `environment()`**:** Return the environment variables set for this content.

Usage:

```
Content$environment()
```

Method `environment_set()`**:** Adjust the environment variables set for this content.

Usage:

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

Arguments:

`...` Environment variable names and values. Use `NA` as the value to unset variables.

Method `environment_all()`**:** Overwrite the environment variables set for this content.

Usage:

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

Arguments:

`...` Environment variable names and values.

Method `deploy()`**:** Deploy this content

Usage:

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

Arguments:

`bundle_id` Target bundle identifier.

Method `repository()`**:** Get Git repository details

Usage:

```
Content$repository()
```

Returns: NULL if no repo is set, otherwise a list with fields:

- repository
- branch
- directory
- polling
- last_error
- last_known_commit

Method `repo_enable()`**:** Adjust Git polling.

Usage:

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

Arguments:

`polling` Polling enabled.

Method `repo_set()`**:** Adjust Git repository

Usage:

```
Content$repo_set(repository, branch = "main", directory = ".", polling = FALSE)
```

Arguments:

`repository` Git repository URL
`branch` Git repository branch
`directory` Git repository directory
`polling` Whether to check for updates

Method `packages()`**:** Get package dependencies

Usage:

```
Content$packages()
```

Method `print()`**:** Print this object.

Usage:

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

Arguments:

... Unused.

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`, `Variant`, `VariantSchedule`, `VariantTask`

`ContentTask` *ContentTask*

Description

ContentTask

ContentTask

Details

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

Super class

`connectapi::Content` -> `ContentTask`

Public fields

`task` The task.

`data` The task data.

Methods

Public methods:

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

Method `new()`**:** Initialize this task.

Usage:

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

Arguments:

`connect` The `Connect` instance.

`content` The `Content` instance.

`task` The task data.

Method `get_task()`**:** Return the underlying task.

Usage:

```
ContentTask$get_task()
```

Method `add_data()`**:** Set the data.

Usage:

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

Arguments:

`data` The data.

Method `get_data()`**:** Get the data.

Usage:

```
ContentTask$get_data()
```

Method `print()`**:** Print this object.

Usage:

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

Arguments:

`...` Unused.

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`, `Variant`, `VariantSchedule`, `VariantTask`

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: content_item(), content_title(), content_update(), create_random_name(), dashboard_url(), delete_thumbnail(), delete_vanity_url(), deploy_repo(), get_associations(), get_bundles(), get_environment(), get_jobs(), get_log(), get_thumbnail(), get_vanity_url(), git, has_thumbnail(), lock_content(), permissions, search_content(), set_integrations(), set_run_as(), set_thumbnail(), set_vanity_url(), swap_vanity_urls(), terminate_jobs(), 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: content_delete(), content_title(), content_update(), create_random_name(), dashboard_url(), delete_thumbnail(), delete_vanity_url(), deploy_repo(), get_associations(), get_bundles(), get_environment(), get_jobs(), get_log(), get_thumbnail(), get_vanity_url(), git, has_thumbnail(), lock_content(), permissions, search_content(), set_integrations(), set_run_as(), set_thumbnail(), set_vanity_url(), swap_vanity_urls(), terminate_jobs(), 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_render` *Render a content item.*

Description

Submit a request to render a content item. Once submitted, the server runs an asynchronous process to render the content. This might be useful if content needs to be updated after its source data has changed, especially if this doesn't happen on a regular schedule.

Only valid for rendered content (e.g., most Quarto documents, Jupyter notebooks, R Markdown reports).

Usage

```
content_render(content, variant_key = NULL)
```

Arguments

`content`	The content item you wish to render.
`variant_key`	If a variant key is provided, render that variant. Otherwise, render the default variant.

Value

A VariantTask object that can be used to track completion of the render.

Examples

```
## Not run:
client <- connect()
item <- content_item(client, "951bf3ad-82d0-4bca-bba8-9b27e35c49fa")
task <- content_render(item)
poll_task(task)

## End(Not run)
```

`content_restart` *Restart a content item.*

Description

Submit a request to restart a content item. Once submitted, the server performs an asynchronous request to kill all processes associated with the content item, starting new processes as needed. This might be useful if the application relies on data that is loaded at startup, or if its memory usage has grown over time.

Note that users interacting with certain types of applications may have their workflows interrupted.

Only valid for interactive content (e.g., applications, APIs).

Usage

```
content_restart(content)
```

Arguments

content	The content item you wish to restart.

Examples

```
## Not run:
client <- connect()
item <- content_item(client, "8f37d6e0-3395-4a2c-aa6a-d7f2fe1babd0")
content_restart(item)

## End(Not run)
```

---

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: content_delete(), content_item(), content_update(), create_random_name(), dashboard_url(), delete_thumbnail(), delete_vanity_url(), deploy_repo(), get_associations(), get_bundles(), get_environment(), get_jobs(), get_log(), get_thumbnail(), get_vanity_url(), git, has_thumbnail(), lock_content(), permissions, search_content(), set_integrations(), set_run_as(), set_thumbnail(), set_vanity_url(), swap_vanity_urls(), terminate_jobs(), 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: `content_delete()`, `content_item()`, `content_title()`, `create_random_name()`, `dashboard_url()`, `delete_thumbnail()`, `delete_vanity_url()`, `deploy_repo()`, `get_associations()`, `get_bundles()`, `get_environment()`, `get_jobs()`, `get_log()`, `get_thumbnail()`, `get_vanity_url()`, `git`, `has_thumbnail()`, `lock_content()`, `permissions`, `search_content()`, `set_integrations()`, `set_run_as()`, `set_thumbnail()`, `set_vanity_url()`, `swap_vanity_urls()`, `terminate_jobs()`, `verify_content_name()`

`create_integration` *Create an OAuth integration*

Description

Creates a new OAuth integration on the Posit Connect server. OAuth integrations allow content to access external resources using OAuth credentials.

You must have administrator privileges to perform this action.

See the Posit Connect documentation on OAuth integrations for more information.

Usage

```
create_integration(client, name, description = NULL, template, config)
```

Arguments

`client`	A `Connect` R6 client object.
`name`	A descriptive name to identify the integration.
`description`	Optional, default `NULL`. A brief description of the integration.
`template`	The template to use to configure this integration (e.g., "custom", "github", "google", "connect").
`config`	A list containing the configuration for the integration. The required fields vary depending on the template selected.

Value

A `connect_integration` object representing the newly created integration. See `get_integration()` for details on the returned object.

See Also

`get_integrations()`, `get_integration()`, `update_integration()`, `delete_integration()`

Other oauth integration functions: `delete_integration()`, `get_associations()`, `get_integration()`, `get_integrations()`, `set_integrations()`, `update_integration()`

Examples

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

# Create a GitHub OAuth integration
github_integration <- create_integration(
  client,
  name = "GitHub Integration",
  description = "Integration with GitHub for OAuth access",
  template = "github",
```

```
  config = list(
    client_id = "your-client-id",
    client_secret = "your-client-secret"
  )
)

# Create a custom OAuth integration
custom_integration <- create_integration(
  client,
  name = "Custom API Integration",
  description = "Integration with our custom API",
  template = "custom",
  config = list(
    auth_mode = "Confidential",
    auth_type = "Viewer",
    authorization_uri = "https://api.example.com/oauth/authorize",
    client_id = "your-client-id",
    client_secret = "your-client-secret",
    token_uri = "https://api.example.com/oauth/token"
  )
)

## End(Not run)
```

`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: `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `dashboard_url()`, `delete_thumbnail()`, `delete_vanity_url()`, `deploy_repo()`, `get_associations()`,

get_bundles(), get_environment(), get_jobs(), get_log(), get_thumbnail(), get_vanity_url(), git, has_thumbnail(), lock_content(), permissions, search_content(), set_integrations(), set_run_as(), set_thumbnail(), set_vanity_url(), swap_vanity_urls(), terminate_jobs(), 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: content_delete(), content_item(), content_title(), content_update(), create_random_name(), delete_thumbnail(), delete_vanity_url(), deploy_repo(), get_associations(), get_bundles(), get_environment(), get_jobs(), get_log(), get_thumbnail(), get_vanity_url(), git, has_thumbnail(), lock_content(), permissions, search_content(), set_integrations(), set_run_as(), set_thumbnail(), set_vanity_url(), swap_vanity_urls(), terminate_jobs(), verify_content_name()

`delete_integration` *Delete an OAuth integration*

Description

Deletes an OAuth integration from the Posit Connect server. This permanently removes the integration and any associated content associations.

You must have administrator privileges to perform this action.

See the Posit Connect documentation on OAuth integrations for more information.

Usage

```
delete_integration(integration)
```

Arguments

integration	A connect_integration object (as returned by get_integrations(), get_integration(), or create_integration()).

Value

Returns NULL invisibly if successful.

See Also

get_integrations(), get_integration(), create_integration(), update_integration()

Other oauth integration functions: create_integration(), get_associations(), get_integration(), get_integrations(), set_integrations(), update_integration()

Examples

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

# Get an integration to delete
integration <- get_integration(client, "your-integration-guid")

# Delete the integration
delete_integration(integration)

## End(Not run)
```

---

delete_runtime_cache *Delete a runtime cache*

Description

Delete a runtime cache from a Connect server. Requires Administrator privileges.

Usage

```
delete_runtime_cache(
  client,
  language,
  version,
  image_name = "Local",
  dry_run = FALSE
)
```

Arguments

client	A Connect object.
language	The language of the cache, either "R" or "Python".
version	The version of the cache, e.g. "4.3.3".
image_name	Optional. The name of the off-host execution image for the cache, or "Local" (the default) for native execution caches.
dry_run	Optional, default FALSE. If true, perform a dry run of the deletion.

Value

A Task object representing the deletion task. If dry_run is TRUE, returns NULL or throws an error if the deletion would fail.

See Also

get_runtime_caches()

Other server management functions: get_runtime_caches()

Examples

```
## Not run:
client <- connect()
task <- delete_runtime_cache(client, "R", "4.3.3")
poll_task(task)

## End(Not run)
```

---

delete_thumbnail *Delete content item thumbnail*

Description

Delete the thumbnail from a content item on Connect.

Usage

```
delete_thumbnail(content)
```

Arguments

content	A content item.

Value

The content item (invisibly).

See Also

Other thumbnail functions: get_thumbnail(), has_thumbnail(), set_thumbnail()

Other content functions: content_delete(), content_item(), content_title(), content_update(), create_random_name(), dashboard_url(), delete_vanity_url(), deploy_repo(), get_associations(), get_bundles(), get_environment(), get_jobs(), get_log(), get_thumbnail(), get_vanity_url(), git, has_thumbnail(), lock_content(), permissions, search_content(), set_integrations(), set_run_as(), set_thumbnail(), set_vanity_url(), swap_vanity_urls(), terminate_jobs(), verify_content_name()

Examples

```
## Not run:
client <- connect()
item <- content_item(client, "8f37d6e0-3395-4a2c-aa6a-d7f2fe1babd0")
thumbnail <- get_thumbnail(item)

## End(Not run)
```

---

delete_vanity_url *Delete the Vanity URL*

Description

Delete the vanity URL for a piece of content.

Usage

```
delete_vanity_url(content)
```

Arguments

content	A Content object

See Also

Other content functions: content_delete(), content_item(), content_title(), content_update(), create_random_name(), dashboard_url(), delete_thumbnail(), deploy_repo(), get_associations(), get_bundles(), get_environment(), get_jobs(), get_log(), get_thumbnail(), get_vanity_url(), git, has_thumbnail(), lock_content(), permissions, search_content(), set_integrations(), set_run_as(), set_thumbnail(), set_vanity_url(), swap_vanity_urls(), terminate_jobs(), 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)


client <- connect()
bnd <- bundle_path(system.file("tests/testthat/examples/static.tar.gz", package = "connectapi"))
deploy(client, bnd)
```

---

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: content_delete(), content_item(), content_title(), content_update(), create_random_name(), dashboard_url(), delete_thumbnail(), delete_vanity_url(), get_associations(), get_bundles(), get_environment(), get_jobs(), get_log(), get_thumbnail(), get_vanity_url(), git, has_thumbnail(), lock_content(), permissions, search_content(), set_integrations(), set_run_as(), set_thumbnail(), set_vanity_url(), swap_vanity_urls(), terminate_jobs(), 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

Environment

Environment

Details

An R6 class that represents a Content's Environment Variables

Super class

connectapi::Content -> Environment

Public fields

env_raw The (raw) set of environment variables.

env_vars The set of environment variables.

Methods

Public methods:

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

Method `new()`**:** Initialize this set of environment variables.

Usage:

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

Arguments:

`connect` The `Connect` instance.

`content` The `Content` instance.

Method `environment()`**:** Fetch the set of environment variables.

Usage:

```
Environment$environment()
```

Method `environment_set()`**:** Update the set of environment variables.

Usage:

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

Arguments:

`...` Environment variable names and values.

Method `environment_all()`**:** Overwrite the set of environment variables.

Usage:

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

Arguments:

`...` Environment variable names and values.

Method `env_refresh()`**:** Fetch the set o environment variables.

Usage:

```
Environment$env_refresh()
```

Method `print()`**:** Print this object.

Usage:

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

Arguments:

`...` Unused.

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`, `Content`, `ContentTask`, `PositConnect`, `Task`, `Vanity`, `Variant`, `VariantSchedule`, `VariantTask`

`get_associations` *Get OAuth associations for a piece of content*

Description

Given a `Content` object, retrieves a list of its OAuth associations. An association contains a content GUID and an association GUID, and indicates that the integration can be used by the content when it runs.

Usage

```
get_associations(x)
```

Arguments

`x`	A `Content` object

Value

A list of OAuth integration associations. Each association includes details such as:

- `app_guid`: The content item's GUID (deprecated, use `content_guid` instead).
- `content_guid`: The content item's GUID.
- `oauth_integration_guid`: The GUID of the OAuth integration.
- `oauth_integration_name`: The name of the OAuth integration.
- `oauth_integration_description`: A description of the OAuth integration.
- `oauth_integration_template`: The template used for this OAuth integration.
- `oauth_integration_auth_type`: The authentication type (e.g., "Viewer" or "Service Account").
- `created_time`: The timestamp when the association was created.

See Also

set_integrations(), get_integrations(), get_integration()

Other oauth integration functions: create_integration(), delete_integration(), get_integration(), get_integrations(), set_integrations(), update_integration()

Other content functions: content_delete(), content_item(), content_title(), content_update(), create_random_name(), dashboard_url(), delete_thumbnail(), delete_vanity_url(), deploy_repo(), get_bundles(), get_environment(), get_jobs(), get_log(), get_thumbnail(), get_vanity_url(), git, has_thumbnail(), lock_content(), permissions, search_content(), set_integrations(), set_run_as(), set_thumbnail(), set_vanity_url(), swap_vanity_urls(), terminate_jobs(), verify_content_name()

Examples

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

# Get OAuth associations for an app.
my_app <- content_item(client, "12345678-90ab-cdef-1234-567890abcdef")
my_app_associations <- get_associations(my_app)

# Given those associations, retrieve the integrations themselves.
my_app_integrations <- purrr::map(
  my_app_associations,
  ~ get_integration(client, .x$oauth_integration_guid)
)

## End(Not run)
```

---

get_audit_logs *Get Audit Logs from Posit Connect Server*

Description

Get Audit Logs from Posit Connect Server

Usage

```
get_audit_logs(src, limit = 500, 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/#get-/v1/audit_logs 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_aws_content_credentials

Obtain AWS credentials for your content.

Description

Obtain AWS credentials for your content.

Usage

```
get_aws_content_credentials(
  connect,
  content_session_token = NULL,
  audience = NULL
)
```

Arguments

connect	A Connect R6 object.
content_session_token	Optional. The content session token. This token can only be obtained when the content is running on a Connect server. The token identifies the service account integration previously configured by the publisher on the Connect server. Defaults to the value from the environment variable: CONNECT_CONTENT_SESSION_TOKEN
audience	Optional. The GUID of an OAuth integration associated with this piece of content.

Details

Please see https://docs.posit.co/connect/user/oauth-integrations/#obtaining-service-account-aws-credentials for more information. See the example below of using this function with paws to access S3. Any library that allows you to pass AWS credentials will be able to utilize the credentials returned from this function call.

Value

The AWS credentials as a list with fields named access_key_id, secret_access_key, session_token, and expiration.

Examples

```
## Not run:
library(connectapi)
library(paws)

client <- connect()
# Pulls the content session token from the environment
# when deployed into Connect.
aws_credentials <- get_aws_content_credentials(client)

# Create S3 client with AWS credentials from Connect
svc <- paws::s3(
  credentials = list(
    creds = list(
      access_key_id = aws_credentials$access_key_id,
      secret_access_key = aws_credentials$secret_access_key,
      session_token = aws_credentials$session_token
    )
  )
```

```
)

# Get object from S3
obj <- svc$get_object(
  Bucket = "my-bucket",
  Key = "my-data.csv"
)

## End(Not run)
```

`get_aws_credentials` *Obtain a visitor's AWS credentials*

Description

Obtain a visitor's AWS credentials

Usage

```
get_aws_credentials(connect, user_session_token, audience = NULL)
```

Arguments

`connect`	A Connect R6 object.
`user_session_token`	The content visitor's session token. This token can only be obtained when the content is running on a Connect server. The token identifies the user who is viewing the content interactively on the Connect server. Read this value from the HTTP header: `Posit-Connect-User-Session-Token`
`audience`	Optional. The GUID of an OAuth integration associated with this piece of content.

Details

Please see https://docs.posit.co/connect/user/oauth-integrations/#obtaining-service-account-aws-credentials for more information. See the example below of using this function in a Plumber API with paws to access S3. Any library that allows you to pass AWS credentials will be able to utilize the credentials returned from this function call.

Value

The AWS credentials as a list with fields named `access_key_id`, `secret_access_key`, `session_token`, and `expiration`.

Examples

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

#* @get /do
function(req) {
  user_session_token <- req$HTTP_POSIT_CONNECT_USER_SESSION_TOKEN
  aws_credentials <- get_aws_credentials(client, user_session_token)

  # Create S3 client with AWS credentials from Connect
  svc <- paws::s3(
    credentials = list(
      creds = list(
        access_key_id = aws_credentials$access_key_id,
        secret_access_key = aws_credentials$secret_access_key,
        session_token = aws_credentials$session_token
      )
    )
  )

  # Get object from S3
  obj <- svc$get_object(
    Bucket = "my-bucket",
    Key = "my-data.csv"
  )

  "done"
}

## End(Not run)
```

`get_bundles` *Get Bundles*

Description

Lists bundles for a content item

Usage

```
get_bundles(content)

delete_bundle(content, bundle_id)
```

Arguments

content	A R6 Content item, as returned by content_item()
bundle_id	A specific bundle ID for a content item

See Also

Other content functions: content_delete(), content_item(), content_title(), content_update(), create_random_name(), dashboard_url(), delete_thumbnail(), delete_vanity_url(), deploy_repo(), get_associations(), get_environment(), get_jobs(), get_log(), get_thumbnail(), get_vanity_url(), git, has_thumbnail(), lock_content(), permissions, search_content(), set_integrations(), set_run_as(), set_thumbnail(), set_vanity_url(), swap_vanity_urls(), terminate_jobs(), verify_content_name()

Other content functions: content_delete(), content_item(), content_title(), content_update(), create_random_name(), dashboard_url(), delete_thumbnail(), delete_vanity_url(), deploy_repo(), get_associations(), get_environment(), get_jobs(), get_log(), get_thumbnail(), get_vanity_url(), git, has_thumbnail(), lock_content(), permissions, search_content(), set_integrations(), set_run_as(), set_thumbnail(), set_vanity_url(), swap_vanity_urls(), terminate_jobs(), 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. It may have a value of `all`, `logged_in` or `acl`. 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.
- `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` setting 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 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.
- vanity_url: The vanity URL associated with this content item.
- id: The internal numeric identifier of this content item.
- tags: Tags associated with this content item. Each entry is a list with the following fields:
  - id: The identifier for the tag.
  - name: The name of the tag.
  - parent_id: The identifier for the parent tag. Null if the tag is a top-level tag.
  - created_time: The timestamp (RFC3339) indicating when the tag was created.
  - updated_time: The timestamp (RFC3339) indicating when the tag was last updated.
- owner: Basic details about the owner of this content item. Each entry is a list with the following fields:
  - guid: The user's GUID, or unique identifier, in UUID RFC4122 format.

- username: The user's username.
- first_name: The user's first name.
- last_name: The user's last name.

Examples

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

get_content(client)

## End(Not run)
```

`get_content_packages` *Package dependencies for a content item*

Description

Get a data frame of package dependencies used by a content item.

Usage

```
get_content_packages(content)
```

Arguments

content	A content item

Value

A data frame with the following columns:

- language : Language ecosystem the package belongs to (r or python)
- name: The package name
- version: The package version
- hash: For R packages, the package DESCRIPTION hash

See Also

Other packages functions: get_packages()

Examples

```
## Not run:
client <- connect()
item <- content_item(client, "951bf3ad-82d0-4bca-bba8-9b27e35c49fa")
packages <- get_content_packages(item)

## 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: content_delete(), content_item(), content_title(), content_update(), create_random_name(), dashboard_url(), delete_thumbnail(), delete_vanity_url(), deploy_repo(), get_associations(), get_bundles(), get_jobs(), get_log(), get_thumbnail(), get_vanity_url(), git, has_thumbnail(), lock_content(), permissions, search_content(), set_integrations(), set_run_as(), set_thumbnail(), set_vanity_url(), swap_vanity_urls(), terminate_jobs(), 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 = 500, prefix = NULL, limit = Inf)
```

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 number of groups to retrieve before paging stops. Default is to return all results; however, for Connect server versions older than 2025.04.0, limit is capped at 500 when prefix is provided.

Details

Please see https://docs.posit.co/connect/api/#get-/v1/groups 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.

See Also

Other groups functions: get_group_content(), get_group_members()

Examples

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

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

## End(Not run)
```

`get_group_content` *Get content access permissions for a group or groups*

Description

Get content access permissions for a group or groups

Usage

```
get_group_content(src, guids)
```

Arguments

`src`	A Connect client object
`guids`	A character vector of group guids

Value

A tibble with the following columns:

- `group_guid`: The group's GUID
- `group_name`: The group's name
- `content_guid`: The content item's GUID
- `content_name`: The content item's name
- `content_title`: The content item's title
- `access_type`: The access type of the content item ("all", "logged_in", or "acl")
- `role`: The access type that members of the group have to the content item, "publisher" or "viewer".

See Also

Other groups functions: `get_group_members()`, `get_groups()`

Examples

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

# Get a data frame of groups
groups <- get_groups(client)

# Get permissions for a single group by passing in the corresponding row.
get_group_content(client, groups[1, "guid"])
dplyr::filter(groups, name = "research_scientists") %>%
  dplyr::pull(guid) %>%
  get_group_content(client, .)

# Get permissions for all groups by passing in all group guids.
get_group_content(client, groups$guid)

## 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`	A Connect client object
`guid`	A group GUID identifier

Details

Please see https://docs.posit.co/connect/api/#get-/v1/groups/-group_guid-/members 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

See Also

Other groups functions: `get_group_content()`, `get_groups()`

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_integration` *Get the details of an OAuth integration*

Description

Given the GUID of an OAuth integration available on a Connect server, retrieve its details. You must have administrator or publisher privileges to perform this action.

Usage

```
get_integration(client, guid)
```

Arguments

client	A Connect R6 client object.
guid	The GUID of an integration available on the Connect server.

Value

A connect_integration object representing an OAuth integration, which has the following fields:

- id: The internal identifier of this OAuth integration.
- guid: The GUID of this OAuth integration.
- created_time: The timestamp (RFC3339) indicating when this integration was created.
- updated_time: The timestamp (RFC3339) indicating when this integration was last updated
- name: A descriptive name to identify the OAuth integration.
- description: A brief text to describe the OAuth integration.
- template: The template used to configure this OAuth integration.
- auth_type: The authentication type indicates which OAuth flow is used by this integration.
- config: A list with the OAuth integration configuration. Fields differ between integrations.

See Also

get_integrations(), get_associations(), set_integrations()

Other oauth integration functions: create_integration(), delete_integration(), get_associations(), get_integrations(), set_integrations(), update_integration()

Examples

```
## Not run:
client <- connect()
x <- get_integration(client, guid)

## End(Not run)
```

---

get_integrations *Get OAuth integrations*

Description

Retrieve OAuth integrations either from the Connect server or associated with a specific content item.

If x is a Connect object, this function lists all OAuth integrations on the server. If x is a Content object, it returns the integrations associated with that content item.

You must have administrator or publisher privileges to use this function.

Usage

```
get_integrations(x)
```

Arguments

x	A Connect or Content R6 object.

Value

A list of class connect_integration_list, where each element is a connect_integration object with the following fields. (Raw API fields are character strings unless noted otherwise):

- id: The internal identifier of this OAuth integration.
- guid: The GUID of this OAuth integration.
- created_time: Timestamp (RFC3339) when the integration was created.
- updated_time: Timestamp (RFC3339) when the integration was last updated.
- name: A descriptive name.
- description: A brief description.
- template: The template used to configure the integration.
- auth_type: The OAuth flow used.
- config: A list with integration-specific config fields.

Use as.data.frame() or tibble::as_tibble() to convert the result to a data frame with parsed types.

See Also

get_integration(), set_integrations(), get_associations()

Other oauth integration functions: create_integration(), delete_integration(), get_associations(), get_integration(), set_integrations(), update_integration()

Examples

```
## Not run:
# From a Connect client
client <- connect()
integrations <- get_integrations(client)

# Filter or update specific ones
github_integration <- purrr::keep(integrations, \(x) x$template == "github")[[1]]

json_payload <- jsonlite::toJSON(list(
  description = "Updated Description",
  config = list(client_secret = "new-secret")
), auto_unbox = TRUE)

client$PATCH(
  paste0("v1/oauth/integrations/", github_integration$guid),
```

```
  body = json_payload
)

# From a Content item
content <- content_item(client, "12345678-90ab-cdef-1234-567890abcdef")
content_integrations <- get_integrations(content)

# Filter content integrations
snowflake_integrations <- purrr::keep(content_integrations, ~ .x$template == "snowflake")

## End(Not run)
```

`get_jobs` *Get Jobs*

Description

Retrieve details about server processes associated with a `content_item`, such as a FastAPI app or a Quarto render.

Usage

```
get_jobs(content)

get_job_list(content)
```

Arguments

`content`	A Content object, as returned by `content_item()`

Details

Note that Connect versions below 2022.10.0 use a legacy endpoint, and will not return the complete set of information provided by newer versions.

`get_jobs()` returns job data as a data frame, whereas `get_jobs_list()` returns job data in a list.

You might get job data as a data frame if you want to perform some calculations about job data (e.g. counting server processes over time), or if you want to filter jobs to find a specific key.

The objects in list returned by `get_jobs_list()` are useful if you want to take an action on a job, such as getting its process log with `get_log()`.

Value

- `get_jobs()`: A data frame with a row representing each job.
- `get_job_list()`: A list with each element representing a job.

Jobs contain the following fields:

- `id`: The job identifier.
- `ppid`: The job's parent process identifier (see Note 1).
- `pid`: The job's process identifier.
- `key`: The job's unique key identifier.
- `remote_id`: The job's identifier for off-host execution configurations (see Note 1).
- `app_id`: The job's parent content identifier; deprecated in favor of `content_id`.
- `app_guid`: The job's parent content GUID; deprecated in favor of `content_guid`.
- `content_id`: The job's parent content identifier.
- `content_guid`: The job's parent content GUID.
- `variant_id`: The identifier of the variant owning this job.
- `bundle_id`: The identifier of a content bundle linked to this job.
- `start_time`: The timestamp (RFC3339) indicating when this job started.
- `end_time`: The timestamp (RFC3339) indicating when this job finished.
- `last_heartbeat_time`: The timestamp (RFC3339) indicating the last time this job was observed to be running (see Note 1).
- `queued_time`: The timestamp (RFC3339) indicating when this job was added to the queue to be processed. Only scheduled reports will present a value for this field (see Note 1).
- `queue_name`: The name of the queue which processes the job. Only scheduled reports will present a value for this field (see Note 1).
- `tag`: A tag to identify the nature of the job.
- `exit_code`: The job's exit code. Present only when job is finished.
- `status`: The current status of the job. On Connect 2022.10.0 and newer, one of Active: 0, Finished: 1, Finalized: 2; on earlier versions, Active: 0, otherwise `NA`.
- `hostname`: The name of the node which processes the job.
- `cluster`: The location where this content runs. Content running on the same server as Connect will have either a null value or the string Local. Gives the name of the cluster when run external to the Connect host (see Note 1).
- `image`: The location where this content runs. Content running on the same server as Connect will have either a null value or the string Local. References the name of the target image when content runs in a clustered environment such as Kubernetes (see Note 1).
- `run_as`: The UNIX user that executed this job.

Note

1. On Connect instances earlier than 2022.10.0, these fields will contain `NA` values.

See Also

Other job functions: `get_log()`, `terminate_jobs()`

Other content functions: `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url()`, `delete_thumbnail()`, `delete_vanity_url()`, `deploy_repo()`, `get_associations()`, `get_bundles()`, `get_environment()`, `get_log()`, `get_thumbnail()`, `get_vanity_url()`, `git`, `has_thumbnail()`, `lock_content()`, `permissions`, `search_content()`, `set_integrations()`, `set_run_as()`, `set_thumbnail()`, `set_vanity_url()`, `swap_vanity_urls()`, `terminate_jobs()`, `verify_content_name()`

Examples

```
## Not run:
client <- connect()
item <- content_item(client, "951bf3ad-82d0-4bca-bba8-9b27e35c49fa")
jobs <- get_jobs(item)
job_list <- get_job_list(item)

## End(Not run)
```

`get_log` *Get Job Log*

Description

Get the log output for a job. Requires Connect 2022.10.0 or newer.

Usage

```
get_log(job, max_log_lines = NULL)
```

Arguments

`job`	A job, represented by an element from the list returned by `get_job_list()`.
`max_log_lines`	Optional. An integer indicating the maximum number of log lines to return. If `NULL` (default), Connect returns a maximum of 5000 lines.

Details

Note: The output of `get_jobs()` cannot be used with `get_log()`. Please use an object from the list returned by `get_job_list()`.

Value

A data frame with the requested log. Each row represents an entry.

- `source`: `stdout` or `stderr`
- `timestamp`: The time of the entry.
- `data`: The logged text.

See Also

Other job functions: `get_jobs()`, `terminate_jobs()`

Other content functions: `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url()`, `delete_thumbnail()`, `delete_vanity_url()`, `deploy_repo()`, `get_associations()`, `get_bundles()`, `get_environment()`, `get_jobs()`, `get_thumbnail()`, `get_vanity_url()`, `git`, `has_thumbnail()`, `lock_content()`, `permissions`, `search_content()`, `set_integrations()`, `set_run_as()`, `set_thumbnail()`, `set_vanity_url()`, `swap_vanity_urls()`, `terminate_jobs()`, `verify_content_name()`

Examples

```
## Not run:
client <- connect()
item <- content_item(client, "951bf3ad-82d0-4bca-bba8-9b27e35c49fa")
jobs <- get_job_list(item)
log <- get_log(jobs[[1]])

## End(Not run)
```

`get_oauth_content_credentials` *Perform an OAuth credential exchange to obtain a content-specific OAuth access token.*

Description

Perform an OAuth credential exchange to obtain a content-specific OAuth access token.

Usage

```
get_oauth_content_credentials(
  connect,
  content_session_token = NULL,
  requested_token_type = NULL,
  audience = NULL
)
```

Arguments

`connect`	A Connect R6 object.
`content_session_token`	Optional. The content session token. This token can only be obtained when the content is running on a Connect server. The token identifies the service account integration previously configured by the publisher on the Connect server. Defaults to the value from the environment variable: `CONNECT_CONTENT_SESSION_TOKEN`
`requested_token_type`	Optional. The requested token type. If unset, will default to `urn:ietf:params:oauth:token-type:acce` Otherwise, this can be set to `urn:ietf:params:aws:token-type:credentials` for AWS integrations or `urn:posit:connect:api-key` for Connect API Key integrations.
`audience`	Optional. The GUID of an OAuth integration associated with this piece of content.

Details

Please see https://docs.posit.co/connect/user/oauth-integrations/#obtaining-a-service-account-oauth-access-token for more information.

Value

The OAuth credential exchange response.

See Also

`get_integrations()`, `get_oauth_credentials()`

Examples

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

#* @get /do
function(req) {
  credentials <- get_oauth_content_credentials(client)

  # ... do something with `credentials$access_token` ...

  "done"
}

## End(Not run)
```

---

get_oauth_credentials *Perform an OAuth credential exchange to obtain a visitor's OAuth access token.*

Description

Perform an OAuth credential exchange to obtain a visitor's OAuth access token.

Usage

```
get_oauth_credentials(
  connect,
  user_session_token,
  requested_token_type = NULL,
  audience = NULL
)
```

Arguments

connect	A Connect R6 object.
user_session_token	The content visitor's session token. This token can only be obtained when the content is running on a Connect server. The token identifies the user who is viewing the content interactively on the Connect server. Read this value from the HTTP header: `Posit-Connect-User-Session-Token`
requested_token_type	Optional. The requested token type. If unset, will default to `urn:ietf:params:oauth:token-type:acce` Otherwise, this can be set to `urn:ietf:params:aws:token-type:credentials` for AWS integrations or `urn:posit:connect:api-key` for Connect API Key integrations.
audience	Optional. The GUID of an OAuth integration associated with this piece of content.

Details

Please see https://docs.posit.co/connect/user/oauth-integrations/#obtaining-a-viewer-oauth-access-token for more information.

Value

The OAuth credential exchange response.

See Also

`get_integrations()`, `get_oauth_content_credentials()`

Examples

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

#* @get /do
function(req) {
  user_session_token <- req$HTTP_POSIT_CONNECT_USER_SESSION_TOKEN
  credentials <- get_oauth_credentials(client, user_session_token)

  # ... do something with `credentials$access_token` ...

  "done"
}

## End(Not run)
```

`get_packages` *All package dependencies on the server*

Description

Get a data frame of package dependencies used by all content items on the server.

Usage

```
get_packages(src, name = NULL, page_size = 100000, limit = Inf)
```

Arguments

`src`	A `Connect` client object.
`name`	Optional package name to filter by. Python package are normalized during matching; R package names must match exactly.
`page_size`	Optional. Integer specifying page size for API paging.
`limit`	Optional. Specify the maximum number of records after which to cease paging.

Value

A data frame with the following columns:

- `language`: Language ecosystem the package belongs to (`r` or `python`)
- `language_version`: Version of R or Python used by the content
- `name`: Package name
- `version`: Package version
- `hash`: Package description hash for R packages
- `bundle_id`: Identifier for the bundle that depends on this package
- `content_id`: Numeric identifier for the content that depends on this package
- `content_guid`: The unique identifier of the content item that depends on this package

See Also

Other packages functions: `get_content_packages()`

Examples

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

## End(Not run)
```

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_runtimes *Get available runtimes on server*

Description

Get a table showing available versions of R, Python, Quarto, and Tensorflow on the Connect server.

Usage

```
get_runtimes(client, runtimes = NULL)
```

Arguments

client	A Connect object.
runtimes	Optional. A character vector of runtimes to include. Must be some combination of "r", "python", "quarto", and "tensorflow". Quarto is only supported on Connect >= 2021.08.0, and Tensorflow is only supported on Connect >= 2024.03.0.

Value

A tibble with columns for runtime, version, and cluster_name and image_name. Cluster name and image name are only meaningful on Connect instances running off-host execution.

Examples

```
## Not run:
library(connectapi)
client <- connect()
get_runtimes(client, runtimes = c("r", "python", "tensorflow"))

## End(Not run)
```

---

get_runtime_caches *Get runtime caches*

Description

View the runtime caches on a Connect server. Requires Administrator privileges.

Usage

```
get_runtime_caches(client)
```

Arguments

client	A Connect object.

Value

A tibble of runtime caches on the server, showing language, version and image_name. For Connect servers not using off-host execution, image_name is "Local".

See Also

delete_runtime_cache()

Other server management functions: delete_runtime_cache()

Examples

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

## End(Not run)
```

`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_thumbnail` *Get content item thumbnail*

Description

Download the thumbnail for a content item on Connect to a file on your computer.

Usage

```
get_thumbnail(content, path = NULL)
```

Arguments

`content`	A content item.
`path`	Optional. A path to a file used to write the thumbnail image. If no path is provided, a temporary file with the correct file extension is created.

Value

The path to the downloaded image file, if `content` has a thumbnail; otherwise `NA`.

See Also

Other thumbnail functions: delete_thumbnail(), has_thumbnail(), set_thumbnail()

Other content functions: content_delete(), content_item(), content_title(), content_update(), create_random_name(), dashboard_url(), delete_thumbnail(), delete_vanity_url(), deploy_repo(), get_associations(), get_bundles(), get_environment(), get_jobs(), get_log(), get_vanity_url(), git, has_thumbnail(), lock_content(), permissions, search_content(), set_integrations(), set_run_as(), set_thumbnail(), set_vanity_url(), swap_vanity_urls(), terminate_jobs(), verify_content_name()

Examples

```
## Not run:
client <- connect()
item <- content_item(client, "8f37d6e0-3395-4a2c-aa6a-d7f2fe1babd0")
thumbnail <- get_thumbnail(item)

## End(Not run)
```

---

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` *Get usage information for deployed content*

Description

Retrieve content hits for all available content on the server. Available content depends on the user whose API key is in use. Administrator accounts will receive data for all content on the server. Publishers will receive data for all content they own or collaborate on.

If no date-times are provided, all usage data will be returned.

Usage

```
get_usage(client, content_guid = NULL, from = NULL, to = NULL)
```

Arguments

`client`	A `Connect` R6 client object.
`content_guid`	Optional. A single content GUID or a character vector of GUIDs to filter results. When multiple GUIDs are provided they are collapsed with `"|"`.
`from`	Optional date-time (`POSIXct` or `POSIXlt`). Only records after this time are returned. If not provided, records are returned back to the first record available.
`to`	Optional date-time (`POSIXct` or `POSIXlt`). Only records before this time are returned. If not provided, all records up to the most recent are returned.

Details

The data returned by `get_usage()` includes all content types. For Shiny content, the `timestamp` indicates the *start* of the Shiny session. Additional fields for Shiny and non-Shiny are available respectively from `get_usage_shiny()` and `get_usage_static()`. `get_usage_shiny()` includes a field for the session end time; `get_usage_static()` includes variant, rendering, and bundle identifiers for the visited content.

When possible, however, we recommend using `get_usage()` over `get_usage_static()` or `get_usage_shiny()`, as it is faster and more efficient.

Value

A list of usage records. Each record is a list with all elements as character strings unless otherwise specified.

- `id`: An integer identifier for the hit.
- `user_guid`: The user GUID if the visitor is logged-in, `NULL` for anonymous hits.
- `content_guid`: The GUID of the visited content.
- `timestamp`: The time of the hit in RFC3339 format.
- `data`: A nested list with optional fields:
  - `path`: The request path (if recorded).

- user_agent: The user agent string (if available).

Use as.data.frame() or tibble::as_tibble() to convert to a flat table with parsed types. In the resulting data frame:

- timestamp is parsed to POSIXct.
- path and user_agent are extracted from the nested data field.

By default, as.data.frame() attempts to extract the nested fields using the **tidyr** package. If **tidyr** is not available, or if you want to skip unnesting, call as.data.frame(x, unnest = FALSE) to leave data as a list-column.

See Also

as.data.frame.connect_list_hits(), as_tibble.connect_list_hits()

Examples

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

# Fetch the last 2 days of hits
usage <- get_usage(client, from = Sys.Date() - 2, to = Sys.Date())

# Fetch usage after a specified date and convert to a data frame.
usage <- get_usage(
  client,
  from = as.POSIXct("2025-05-02 12:40:00", tz = "UTC")
)

# Fetch usage for a specific content item
usage <- get_usage(client, content_guid = "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee")

# Fetch all usage
usage <- get_usage(client)

# Convert to tibble or data frame
usage_df <- tibble::as_tibble(usage)

# Skip unnesting if tidyr is not installed
usage_df <- as.data.frame(usage, unnest = FALSE)

## End(Not run)
```

---

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 = 500,
  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/#get-/v1/instrumentation/shiny/usage 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 = 500,
  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/#get-/v1/instrumentation/content/visits 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 = 500,
  prefix = NULL,
  limit = Inf,
  user_role = NULL,
  account_status = NULL
)
```

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
`user_role`	Optionally filter by user role ("administrator", "publisher", "viewer"). Pass in a vector of multiple roles to match any value (boolean OR). When `NULL` (the default), results are not filtered.
`account_status`	Optionally filter by account status ("locked", "licensed", "inactive"). Pass a vector of multiple statuses to match any value (boolean OR). When `NULL` (the default), results are not filtered.

Details

Please see https://docs.posit.co/connect/api/#get-/v1/users 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)

# Get all licensed users
get_users(client, account_status = "licensed")

# Get all users who are administrators or publishers
get_users(client, user_role = c("administrator", "publisher"))

## End(Not run)
```

---

get_vanity_url *Get the Vanity URL*

Description

Get 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: content_delete(), content_item(), content_title(), content_update(), create_random_name(), dashboard_url(), delete_thumbnail(), delete_vanity_url(), deploy_repo(), get_associations(), get_bundles(), get_environment(), get_jobs(), get_log(), get_thumbnail(), git, has_thumbnail(), lock_content(), permissions, search_content(), set_integrations(), set_run_as(), set_thumbnail(), set_vanity_url(), swap_vanity_urls(), terminate_jobs(), verify_content_name()

---

get_vanity_urls *Get all vanity URLs*

Description

Get a table of all vanity URLs on the server. Requires administrator privileges.

Usage

```
get_vanity_urls(client)
```

Arguments

client	A Connect object.

Value

A tibble with columns for content_guid, path, and created_time.

Examples

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

## End(Not run)
```

get_variants *Get Variant*

Description

[Experimental] Work with variants

Usage

```
get_variants(content)

get_variant(content, key)

get_variant_default(content)
```

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_variant_default() 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: `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url()`, `delete_thumbnail()`, `delete_vanity_url()`, `deploy_repo()`, `get_associations()`, `get_bundles()`, `get_environment()`, `get_jobs()`, `get_log()`, `get_thumbnail()`, `get_vanity_url()`, `has_thumbnail()`, `lock_content()`, `permissions`, `search_content()`, `set_integrations()`, `set_run_as()`, `set_thumbnail()`, `set_vanity_url()`, `swap_vanity_urls()`, `terminate_jobs()`, `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, exact = FALSE)
```

Arguments

connect	An 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 groups whose name exactly matches the provided prefix.

Value

The results of creating the groups.

has_thumbnail *Check content item thumbnail*

Description

Check whether a content item has a thumbnail.

Usage

```
has_thumbnail(content)
```

Arguments

content	A content item.

Value

TRUE if the content item has a thumbnail, otherwise FALSE. Throws an error if you do not have permission to view the thumbnail.

See Also

Other thumbnail functions: delete_thumbnail(), get_thumbnail(), set_thumbnail()

Other content functions: content_delete(), content_item(), content_title(), content_update(), create_random_name(), dashboard_url(), delete_thumbnail(), delete_vanity_url(), deploy_repo(), get_associations(), get_bundles(), get_environment(), get_jobs(), get_log(), get_thumbnail(), get_vanity_url(), git, lock_content(), permissions, search_content(), set_integrations(), set_run_as(), set_thumbnail(), set_vanity_url(), swap_vanity_urls(), terminate_jobs(), verify_content_name()

Examples

```
## Not run:
client <- connect()
item <- content_item(client, "8f37d6e0-3395-4a2c-aa6a-d7f2fe1babd0")
has_thumbnail(item)

## End(Not run)
```

---

lock_content *Lock or Unlock Content*

Description

Lock or unlock a content item. When content is locked, all processes are terminated, rendering is disabled, and new bundles cannot be deployed.

Usage

```
lock_content(content, locked_message = "")

unlock_content(content)
```

Arguments

content	An R6 content item
locked_message	Optional. A custom message that is displayed by the content item when locked. It is possible to format this message using Markdown.

Details

lock_content() locks a content item with an optional message displayed to visitors (supports Markdown).

unlock_content() unlocks a content item, reverting the effects of locking.

Value

An R6 content item

See Also

Other content functions: content_delete(), content_item(), content_title(), content_update(), create_random_name(), dashboard_url(), delete_thumbnail(), delete_vanity_url(), deploy_repo(), get_associations(), get_bundles(), get_environment(), get_jobs(), get_log(), get_thumbnail(), get_vanity_url(), git, has_thumbnail(), permissions, search_content(), set_integrations(), set_run_as(), set_thumbnail(), set_vanity_url(), swap_vanity_urls(), terminate_jobs(), verify_content_name()

Examples

```
## Not run:
# Lock content with a message
client <- connect()
content <- content_item(client, "content-guid")
content <- lock_content(content, locked_message = "Ah ah ah! You didn't say the magic word!")

# Lock content without a message
content <- lock_content(content)

# Unlock content
content <- unlock_content(content)

## End(Not run)
```

---

page_cursor *Paging*

Description

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	For page_cursor, the output from an initial response to an API endpoint that uses cursor-based pagination. For page_offset, a request that needs to be paged.
limit	A row limit

Value

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: `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url()`, `delete_thumbnail()`, `delete_vanity_url()`, `deploy_repo()`, `get_associations()`, `get_bundles()`, `get_environment()`, `get_jobs()`, `get_log()`, `get_thumbnail()`, `get_vanity_url()`, `git`, `has_thumbnail()`, `lock_content()`, `search_content()`, `set_integrations()`, `set_run_as()`, `set_thumbnail()`, `set_vanity_url()`, `swap_vanity_urls()`, `terminate_jobs()`, `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',
  api_key = 'mysecretkey')

get_content(client)
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.

Public fields

`server` The base URL of your Posit Connect server.

`api_key` Your Posit Connect API key.

`tags` The initial set of tags.

`tag_map` The initial tag map.

`httr_additions` An initial set of `httr` configuration added to each HTTP call.

`using_auth` Indicates that the API key is added to each HTTP call.

Active bindings

`version` The server version.

`timezones` The server timezones.

Methods

Public methods:

- `Connect$new()`
- `Connect$httr_config()`
- `Connect$print()`
- `Connect$raise_error()`
- `Connect$add_auth()`
- `Connect$api_url()`

- Connect$server_url()
- Connect$request()
- Connect$GET()
- Connect$PUT()
- Connect$HEAD()
- Connect$DELETE()
- Connect$PATCH()
- Connect$POST()
- Connect$me()
- Connect$get_dashboard_url()
- Connect$get_tags()
- Connect$get_tag_id()
- Connect$get_tag_tree()
- Connect$tag_create_safe()
- Connect$tag_create()
- Connect$tag()
- Connect$tag_delete()
- Connect$get_schedule()
- Connect$content_create()
- Connect$content_upload()
- Connect$content_deploy()
- Connect$content()
- Connect$task()
- Connect$set_content_tag()
- Connect$remove_content_tag()
- Connect$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$group_content()
- Connect$inst_content_visits()

- Connect$inst_shiny_usage()
- Connect$procs()
- Connect$repo_account()
- Connect$repo_branches()
- Connect$repo_manifest_dirs()
- Connect$schedules()
- Connect$packages()
- Connect$docs()
- Connect$audit_logs()
- Connect$vanities()
- Connect$server_settings()
- Connect$clone()

Method `new()`**:** Initialize a new connect.

Usage:

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

Arguments:

`server` The base URL of your Posit Connect server.

`api_key` Your Posit Connect API key.

Method `httr_config()`**:** Set additional `httr` configuration that is added to each HTTP call.

Usage:

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

Arguments:

`...` Set of httr configurations.

Method `print()`**:** Print details about this instance.

Usage:

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

Arguments:

`...` Ignored.

Method `raise_error()`**:** Raise an error when the HTTP result is an HTTP error.

Usage:

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

Arguments:

`res` HTTP result.

Method `add_auth()`**:** Returns HTTP authorization headers, or NULL when none are used.

Usage:

```
Connect$add_auth()
```

Method `api_url()`**:** Build a URL relative to the API root

Usage:

```
Connect$api_url(...)
```

Arguments:

`...` path segments

Method `server_url()`**:** Build a URL relative to the server root

Usage:

```
Connect$server_url(...)
```

Arguments:

`...` path segments

Method `request()`**:** General wrapper around `httr` verbs

Usage:

```
Connect$request(method, url, ..., parser = "parsed")
```

Arguments:

`method` HTTP request method

`url` URL to request

`...` Additional arguments passed to the request function

`parser` How the response is parsed. If `NULL`, the `httr_response` will be returned. Otherwise, the argument is forwarded to `httr::content(res, as = parser)`.

Method `GET()`**:** Perform an HTTP GET request of the named API path.

Usage:

```
Connect$GET(path, ..., url = self$api_url(path), parser = "parsed")
```

Arguments:

`path` API path relative to the server's `/__api__` root.

`...` Arguments to `httr::GET()`

`url` Target URL. Default uses `path`, but provide `url` to request a server resource that is not under `/__api__`

`parser` How the response is parsed. If `NULL`, the `httr_response` will be returned. Otherwise, the argument is forwarded to `httr::content(res, as = parser)`.

Method `PUT()`**:** Perform an HTTP PUT request of the named API path.

Usage:

```
Connect$PUT(
  path,
  body = "{}",
  ...,
  url = self$api_url(path),
  encode = "json",
  parser = "parsed"
)
```

Arguments:

path API path relative to the server's /__api__ root.

body The HTTP payload.

... Arguments to httr::PUT()

url Target URL. Default uses path, but provide url to request a server resource that is not under /__api__

encode How the payload is encoded.

parser How the response is parsed. If NULL, the httr_response will be returned. Otherwise, the argument is forwarded to httr::content(res, as = parser).

Method HEAD(): Perform an HTTP HEAD request of the named API path.

Usage:

```
Connect$HEAD(path, ..., url = self$api_url(path))
```

Arguments:

path API path relative to the server's /__api__ root.

... Arguments to httr::HEAD()

url Target URL. Default uses path, but provide url to request a server resource that is not under /__api__ httr::content(res, as = parser).

Method DELETE(): Perform an HTTP DELETE request of the named API path. Returns the HTTP response object.

Usage:

```
Connect$DELETE(path, ..., url = self$api_url(path), parser = NULL)
```

Arguments:

path API path relative to the server's /__api__ root.

... Arguments to httr::DELETE()

url Target URL. Default uses path, but provide url to request a server resource that is not under /__api__

parser How the response is parsed. If NULL, the httr_response will be returned. Otherwise, the argument is forwarded to httr::content(res, as = parser).

Method PATCH(): Perform an HTTP PATCH request of the named API path.

Usage:

```
Connect$PATCH(
  path,
  body = "{}",
  ...,
  url = self$api_url(path),
  encode = "json",
  parser = "parsed"
)
```

Arguments:

path API path relative to the server's /__api__ root.

body The HTTP payload.

... Arguments to httr::PATCH()

`url` Target URL. Default uses `path`, but provide `url` to request a server resource that is not under `/__api__`

`encode` How the payload is encoded.

`parser` How the response is parsed. If `NULL`, the `httr_response` will be returned. Otherwise, the argument is forwarded to `httr::content(res, as = parser)`.

Method `POST()`**:** Perform an HTTP POST request of the named API path.

Usage:

```
Connect$POST(
  path,
  body = "{}",
  ...,
  url = self$api_url(path),
  encode = "json",
  parser = "parsed"
)
```

Arguments:

`path` API path relative to the server's `/__api__` root.

`body` The HTTP payload.

`...` Arguments to `httr::POST()`

`url` Target URL. Default uses `path`, but provide `url` to request a server resource that is not under `/__api__`

`encode` How the payload is encoded.

`parser` How the response is parsed. If `NULL`, the `httr_response` will be returned. Otherwise, the argument is forwarded to `httr::content(res, as = parser)`.

Method `me()`**:** Perform an HTTP GET request of the "me" server endpoint.

Usage:

```
Connect$me()
```

Method `get_dashboard_url()`**:** Return the base URL of the Connect server.

Usage:

```
Connect$get_dashboard_url()
```

Method `get_tags()`**:** Return all tags.

Usage:

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

Arguments:

`use_cache` Indicates that a cached set of tags is used.

Method `get_tag_id()`**:** Get the identifier for the named tag.

Usage:

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

Arguments:

`tagname` The name of the tag.

Method `get_tag_tree()`**:** Get the tag tree.

Usage:

```
Connect$get_tag_tree()
```

Method `tag_create_safe()`**:** Create a tag.

Usage:

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

Arguments:

`name` The tag name.

`parent_id` The parent identifier.

Method `tag_create()`**:** Create a tag.

Usage:

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

Arguments:

`name` The tag name.

`parent_id` The parent identifier.

Method `tag()`**:** Get a tag.

Usage:

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

Arguments:

`id` The tag identifier.

Method `tag_delete()`**:** Delete a tag.

Usage:

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

Arguments:

`id` The tag identifier.

Method `get_schedule()`**:** Get a schedule.

Usage:

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

Arguments:

`schedule_id` The schedule identifier.

Method `content_create()`**:** Create content.

Usage:

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

Arguments:

`name` The content name.

`title` The content title.

`...` Other content fields.

Method `content_upload()`**:** Upload a content bundle.

Usage:

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

Arguments:

`bundle_path` The path to the bundle archive.

`guid` The content GUID.

Method `content_deploy()`**:** Deploy a content bundle.

Usage:

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

Arguments:

`guid` The content GUID.

`bundle_id` The bundle identifier.

Method `content()`**:** Get a content item.

Usage:

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

Arguments:

`guid` The content GUID.

`owner_guid` The target content owner.

`name` The target name.

`include` Additional response fields.

Method `task()`**:** Get a task.

Usage:

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

Arguments:

`task_id` The task identifier.

`first` The initial status position.

`wait` Maximum time to wait for update.

Method `set_content_tag()`**:** Set a tag for a content item.

Usage:

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

Arguments:

`content_id` The content identifier.

`tag_id` The tag identifier.

Method `remove_content_tag()`**:** Remove a tag from a content item.

Usage:

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

Arguments:

`content_id` The content identifier.

`tag_id` The tag identifier.

Method `user()`**:** Get user details.

Usage:

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

Arguments:

`guid` The user GUID.

Method `users()`**:** Get users.

Usage:

```
Connect$users(
  page_number = 1,
  prefix = NULL,
  page_size = 500,
  user_role = NULL,
  account_status = NULL
)
```

Arguments:

`page_number` The page number.

`prefix` The search term.

`page_size` The page size.

`user_role` Filter by user role.

`account_status` Filter by account status.

Method `users_remote()`**:** Get remote users.

Usage:

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

Arguments:

`prefix` The search term.

Method `users_create()`**:** Create a user.

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
)
```

Arguments:

`username` The username.

`email` Email address.

`first_name` First name.

`last_name` Last name.

`password` The password.

`user_must_set_password` Indicates that user sets password on first login.

`user_role` Role for user.

`unique_id` Identifier for user.

Method `users_create_remote()`**:** Create a remote user.

Usage:

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

Arguments:

`temp_ticket` Ticket identifying target remote user.

Method `users_lock()`**:** Lock a user.

Usage:

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

Arguments:

`user_guid` User GUID.

Method `users_unlock()`**:** Unlock a user.

Usage:

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

Arguments:

`user_guid` User GUID.

Method `users_update()`**:** Update a user.

Usage:

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

Arguments:

`user_guid` User GUID.

... User fields.

Method `groups()`**:** Get groups.

Usage:

```
Connect$groups(page_number = 1, prefix = NULL, page_size = 500)
```

Arguments:

`page_number` The page number.
`prefix` The search term.
`page_size` The page size.

Method `group_members()`**:** Get group members.

Usage:

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

Arguments:

`guid` The group GUID.

Method `group_member_add()`**:** Add a group member.

Usage:

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

Arguments:

`group_guid` The group GUID.
`user_guid` The user GUID.

Method `group_member_remove()`**:** Remove a group member.

Usage:

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

Arguments:

`group_guid` The group GUID.
`user_guid` The user GUID.

Method `groups_create()`**:** Create a group.

Usage:

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

Arguments:

`name` The group name.

Method `groups_create_remote()`**:** Create a remote group.

Usage:

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

Arguments:

`temp_ticket` Ticket identifying target remote group.

Method `groups_remote()`**:** Get remote groups.

Usage:

```
Connect$groups_remote(prefix = NULL, limit = 500)
```

Arguments:

`prefix` The search term.

`limit` The maximal result set size.

Method `group_content()`**:** Get content to which a group has access

Usage:

```
Connect$group_content(guid)
```

Arguments:

`guid` The group GUID.

Method `inst_content_visits()`**:** Get (non-interactive) content visits.

Usage:

```
Connect$inst_content_visits(
  content_guid = NULL,
  min_data_version = NULL,
  from = NULL,
  to = NULL,
  limit = 500,
  previous = NULL,
  nxt = NULL,
  asc_order = TRUE
)
```

Arguments:

`content_guid` Content GUID.

`min_data_version` Data version for request.

`from` Start of range.

`to` End of range.

`limit` Result set size.

`previous` Previous item.

`nxt` Next item.

`asc_order` Indicates ascending result order.

Method `inst_shiny_usage()`**:** Get interactive content visits.

Get (non-interactive) content visits.

Usage:

```
Connect$inst_shiny_usage(
  content_guid = NULL,
  min_data_version = NULL,
  from = NULL,
  to = NULL,
  limit = 500,
  previous = NULL,
  nxt = NULL,
  asc_order = TRUE
)
```

Arguments:

`content_guid` Content GUID.

`min_data_version` Data version for request.

`from` Start of range.

`to` End of range.

`limit` Result set size.

`previous` Previous item.

`nxt` Next item.

`asc_order` Indicates ascending result order.

Method `procs()`**:** Get running processes.

Usage:

```
Connect$procs()
```

Method `repo_account()`**:** Determine if Git repository is associated with authorization.

Usage:

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

Arguments:

`host` Repository URL.

Method `repo_branches()`**:** Get Git repository branches.

Usage:

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

Arguments:

`repo` Repository URL.

Method `repo_manifest_dirs()`**:** Get Git repository directories.

Usage:

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

Arguments:

`repo` Repository URL.

`branch` Repository branch.

Method `schedules()`**:** Get schedules.

Usage:

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

Arguments:

`start` Starting time.

`end` Ending time.

`detailed` Indicates detailed schedule information.

Method `packages()`**:** Get packages. This endpoint is paginated.

Usage:

```
Connect$packages(name = NULL, page_number = 1, page_size = 1e+05)
```

Arguments:

`name` The package name to filter by.
`page_number` Page number.
`page_size` Page size, default 100000.

Method `docs()`**:** Get documentation.

Usage:

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

Arguments:

`docs` Named document.
`browse` Open a browser.

Method `audit_logs()`**:** Get auditing.

Usage:

```
Connect$audit_logs(limit = 500, previous = NULL, nxt = NULL, asc_order = TRUE)
```

Arguments:

`limit` Result set size.
`previous` Previous item.
`nxt` Next item.
`asc_order` Indicates ascending result order.

Method `vanities()`**:** Get all vanity URLs

Usage:

```
Connect$vanities()
```

Method `server_settings()`**:** Get 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`, `Content`, `ContentTask`, `Environment`, `Task`, `Vanity`, `Variant`, `VariantSchedule`, `VariantTask`

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

search_content *Search for content on the Connect server*

Description

Search for content on the Connect server

Usage

```
search_content(
  client,
  q = NULL,
  include = "owner,vanity_url",
  page_size = 500,
  limit = Inf,
  ...
)
```

Arguments

client	A Connect object
q	The search query, using the syntax described in the Connect documentation on content search terms
include	Comma-separated character string of values indicating additional details to include in the response. Values can be owner and vanity_url; both are included by default.
page_size	The number of items to fetch per page. Maximum is 500.
limit	Maximum number of items to return overall. Defaults to Inf (all items).
...	Additional query parameters passed to the API for future expansion. Note: If you pass page_number here, it will affect the *starting* page for pagination, but all subsequent pages will still be fetched. This is usually not what you want.

Details

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

Value

A list of Content objects, of class "connect_content_list"

See Also

Other content functions: content_delete(), content_item(), content_title(), content_update(), create_random_name(), dashboard_url(), delete_thumbnail(), delete_vanity_url(), deploy_repo(), get_associations(), get_bundles(), get_environment(), get_jobs(), get_log(), get_thumbnail(), get_vanity_url(), git, has_thumbnail(), lock_content(), permissions, set_integrations(), set_run_as(), set_thumbnail(), set_vanity_url(), swap_vanity_urls(), terminate_jobs(), verify_content_name()

Examples

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

my_content <- search_content(client, q = "owner:@me")

shiny_content <- purrr::keep(my_content, function(x) {
  x$content$app_mode == "rmd-shiny"
})

purrr::map(shiny_content, lock_content)

## End(Not run)
```

`set_integrations` *Set all OAuth integrations for a content item*

Description

Removes all existing OAuth integrations associated with a content item, and creates associations with the integrations provided. You must have administrator or publisher privileges to perform this action.

Usage

```
set_integrations(content, integrations)
```

Arguments

`content`	A `Content` R6 object representing the content item to modify.
`integrations`	The complete set of integrations to be associated with the content. May be a single `connect_integration` object, a list of `connect_integration` objects, or `NULL`. Passing in an empty list or explicitly passing `NULL` will remove all associated integrations from the content.

Value

Invisibly returns `NULL`.

See Also

`get_integrations()`, `get_integration()`, `get_associations()`, `content_item()`

Other oauth integration functions: `create_integration()`, `delete_integration()`, `get_associations()`, `get_integration()`, `get_integrations()`, `update_integration()`

Other content functions: `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url()`, `delete_thumbnail()`, `delete_vanity_url()`, `deploy_repo()`, `get_associations()`, `get_bundles()`, `get_environment()`, `get_jobs()`, `get_log()`, `get_thumbnail()`, `get_vanity_url()`, `git`, `has_thumbnail()`, `lock_content()`, `permissions`, `search_content()`, `set_run_as()`, `set_thumbnail()`, `set_vanity_url()`, `swap_vanity_urls()`, `terminate_jobs()`, `verify_content_name()`

Examples

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

content <- content_item(client, "12345678-90ab-cdef-1234-567890abcdef")

integrations <- get_integrations(client)

# Associate a single integration
```

```
github_integration <- purrr::keep(integrations, \(x) x$template == "github")[[1]]
set_integrations(content, github_integration)

# Associate multiple integrations at once
selected_integrations <- integrations[1:2]
set_integrations(content, selected_integrations)

# Unset integrations
set_integrations(content, NULL)

## End(Not run)
```

`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: content_delete(), content_item(), content_title(), content_update(), create_random_name(), dashboard_url(), delete_thumbnail(), delete_vanity_url(), deploy_repo(), get_associations(), get_bundles(), get_environment(), get_jobs(), get_log(), get_thumbnail(), get_vanity_url(), git, has_thumbnail(), lock_content(), permissions, search_content(), set_integrations(), set_thumbnail(), set_vanity_url(), swap_vanity_urls(), terminate_jobs(), 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	**[Deprecated]** This parameter no longer has any effect due to changes in the Connect API and will be removed in a future release.
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_thumbnail` *Set content item thumbnail*

Description

Set the thumbnail for a content item.

Usage

```
set_thumbnail(content, path)
```

Arguments

`content`	A content item.
`path`	Either a path to a local file or a URL to an image available over HTTP/HTTPS. If `path` is an HTTP or HTTPS URL, the image will first be downloaded.

Value

The content item (invisibly).

See Also

Other thumbnail functions: `delete_thumbnail()`, `get_thumbnail()`, `has_thumbnail()`

Other content functions: `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url()`, `delete_thumbnail()`, `delete_vanity_url()`, `deploy_repo()`, `get_associations()`, `get_bundles()`, `get_environment()`, `get_jobs()`, `get_log()`, `get_thumbnail()`, `get_vanity_url()`, `git`, `has_thumbnail()`, `lock_content()`, `permissions`, `search_content()`, `set_integrations()`, `set_run_as()`, `set_vanity_url()`, `swap_vanity_urls()`, `terminate_jobs()`, `verify_content_name()`

Examples

```
## Not run:
client <- connect()
item <- content_item(client, "8f37d6e0-3395-4a2c-aa6a-d7f2fe1babd0")
set_thumbnail(item, "resources/image.png")

## End(Not run)
```

`set_vanity_url` *Set the Vanity URL*

Description

Set 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: `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url()`, `delete_thumbnail()`, `delete_vanity_url()`, `deploy_repo()`, `get_associations()`, `get_bundles()`, `get_environment()`, `get_jobs()`, `get_log()`, `get_thumbnail()`, `get_vanity_url()`, `git`, `has_thumbnail()`, `lock_content()`, `permissions`, `search_content()`, `set_integrations()`, `set_run_as()`, `set_thumbnail()`, `swap_vanity_urls()`, `terminate_jobs()`, `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_urls *Swap Vanity URLs*

Description

Swap the vanity URLs of two pieces of content.

Usage

```
swap_vanity_urls(content_a, content_b)
```

Arguments

content_a	A Content object
content_b	A Content object

Value

A list of the new vanity URLs for content_a and content_b

See Also

Other content functions: content_delete(), content_item(), content_title(), content_update(), create_random_name(), dashboard_url(), delete_thumbnail(), delete_vanity_url(), deploy_repo(), get_associations(), get_bundles(), get_environment(), get_jobs(), get_log(), get_thumbnail(), get_vanity_url(), git, has_thumbnail(), lock_content(), permissions, search_content(), set_integrations(), set_run_as(), set_thumbnail(), set_vanity_url(), terminate_jobs(), verify_content_name()

Task *Task*

Description

Task

Task

Details

An R6 class that represents a Task

Public fields

connect The Connect instance.

task The task.

data The task data.

Methods

Public methods:

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

Method `new()`**:** Initialize this task.

Usage:

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

Arguments:

`connect` The `Connect` instance.

`task` The task data.

Method `get_task()`**:** Return the underlying task.

Usage:

```
Task$get_task()
```

Method `add_data()`**:** Set the data.

Usage:

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

Arguments:

`data` The data.

Method `get_data()`**:** Get the data.

Usage:

```
Task$get_data()
```

Method `print()`**:** Print this object.

Usage:

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

Arguments:

`...` Unused.

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`, `Content`, `ContentTask`, `Environment`, `PositConnect`, `Vanity`, `Variant`, `VariantSchedule`, `VariantTask`

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

terminate_jobs *Terminate Jobs*

Description

Register a job kill order for one or more jobs associated with a content item. Requires Connect 2022.10.0 or newer.

Usage

```
terminate_jobs(content, keys = NULL)
```

Arguments

`content`	A Content object, as returned by `content_item()`
`keys`	Optional. One or more job keys, which can be obtained using `get_jobs(content)`. If no keys are provided, will terminate all active jobs for the provided content item.

Value

A data frame with the status of each termination request.

- `app_id`: The content item's identifier.
- `app_guid`: The content item's GUID.
- `job_key`: The job key.
- `job_id`: The job's identifier.
- `result`: The result string returned by Connect.
- `code`: An error code, `NA` if the request was successful.
- `error`: An error message, `NA` if the result was successful.

Note that `app_id`, `app_guid`, `job_id`, and `result` are `NA` if the request returns an error.

See Also

Other job functions: `get_jobs()`, `get_log()`

Other content functions: `content_delete()`, `content_item()`, `content_title()`, `content_update()`, `create_random_name()`, `dashboard_url()`, `delete_thumbnail()`, `delete_vanity_url()`, `deploy_repo()`, `get_associations()`, `get_bundles()`, `get_environment()`, `get_jobs()`, `get_log()`, `get_thumbnail()`, `get_vanity_url()`, `git`, `has_thumbnail()`, `lock_content()`, `permissions`, `search_content()`, `set_integrations()`, `set_run_as()`, `set_thumbnail()`, `set_vanity_url()`, `swap_vanity_urls()`, `verify_content_name()`

Examples

```
## Not run:
client <- connect()
item <- content_item(client, "951bf3ad-82d0-4bca-bba8-9b27e35c49fa")
result <- terminate_jobs(item)

## End(Not run)
```

---

update_integration *Update an OAuth integration*

Description

Updates an existing OAuth integration. All fields except `integration` are optional, and are unchanged if not provided.

You must have administrator privileges to perform this action.

See the Posit Connect documentation on OAuth integrations for more information.

Usage

```
update_integration(
  integration,
  name = NULL,
  description = NULL,
  template = NULL,
  config = NULL
)
```

Arguments

integration	A connect_integration object (as returned by get_integrations(), get_integration(), or create_integration()).
name	A new name for the integration.
description	A new description for the integration.
template	The template to use (generally not changed after creation).
config	A list with updated OAuth integration configuration. If NULL (default), the configuration remains unchanged. You can update individual configuration fields without affecting others.

Value

A connect_integration object representing the updated OAuth integration. See get_integration() for details on the returned object.

See Also

get_integrations(), get_integration(), create_integration(), delete_integration()

Other oauth integration functions: create_integration(), delete_integration(), get_associations(), get_integration(), get_integrations(), set_integrations()

Examples

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

# Get an existing integration
integration <- get_integration(client, "your-integration-guid")

# Update the integration's name and description
updated_integration <- update_integration(
  integration,
  name = "Updated GitHub Integration",
  description = "A more descriptive description."
)

# Update only the client secret in the configuration
updated_integration <- update_integration(
  integration,
```

```
  config = list(
    client_secret = "your-new-client-secret"
  )
)

## End(Not run)
```

`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`	An 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

Vanity

Vanity

Details

An R6 class that represents a Vanity URL

Super class

`connectapi::Content` -> `Vanity`

Public fields

`vanity` The vanity.

Methods

Public methods:

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

Method `new()`**:** Initialize this vanity.

Usage:

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

Arguments:

`connect` The `Connect` instance.

`content` The `Content` instance.

`vanity` The vanity data.

Method `get_vanity()`**:** Return the underlying vanity.

Usage:

```
Vanity$get_vanity()
```

Method `print()`**:** Print this object.

Usage:

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

Arguments:

`...` Unused.

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`, `Content`, `ContentTask`, `Environment`, `PositConnect`, `Task`, `Variant`, `VariantSchedule`, `VariantTask`

vanity_is_available *Check to see if a vanity URL is currently in use*

Description

[Experimental]

Usage

```
vanity_is_available(connect, vanity)
```

Arguments

connect	A Connect R6 object
vanity	string of the vanity URL to check

Value

logical indicating if the vanity URL is available.

See Also

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

Variant *Variant*

Description

Variant

Variant

Details

An R6 class that represents a Variant

Super class

connectapi::Content -> Variant

Public fields

key The variant key.

variant The variant.

Methods

Public methods:

Method `get_variant_remote()`**:** Get the underlying variant data.
Get and store the (remote) variant data.

Usage:

```
Variant$get_variant_remote()
```

Method `new()`**:** Initialize this variant.

Usage:

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

Arguments:

`connect` The `Connect` instance.

`content` The `Content` instance.

`key` The variant key.

Method `send_mail()`**:** Mail previously rendered content.

Usage:

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

Arguments:

`to` Targeting.

Method `get_schedule()`**:** Get the (remote) schedule data.

Usage:

```
Variant$get_schedule()
```

Method `get_schedule_remote()`**:** Get the (remote) schedule data.

Usage:

```
Variant$get_schedule_remote()
```

Method `get_subscribers()`**:** Get the subscribers.

Usage:

```
Variant$get_subscribers()
```

Method `remove_subscriber()`**:** Remove a named subscriber.

Usage:

```
Variant$remove_subscriber(guid)
```

Arguments:

`guid` User GUID.

Method `add_subscribers()`**:** Add named subscribers.

Usage:

```
Variant$add_subscribers(guids)
```

Arguments:

`guids` User GUIDs.

Method `render()`**:** Render this variant.

Usage:

```
Variant$render()
```

Method `renderings()`**:** List the renderings of this variant.

Usage:

```
Variant$renderings()
```

Method `update_variant()`**:** Update this variant.

Usage:

```
Variant$update_variant(...)
```

Arguments:

`...` Target fields and values.

Method `jobs()`**:** Jobs for this variant.

Usage:

```
Variant$jobs()
```

Method `get_url()`**:** Return the URL for this variant.

Usage:

```
Variant$get_url()
```

Method `get_url_rev()`**:** Return the URL associated with one rendering for this variant.

Usage:

```
Variant$get_url_rev(rev)
```

Arguments:

`rev` Rendering identifier.

Method `get_dashboard_url()`**:** Return the URL for this variant in the Posit Connect dashboard.

Usage:

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

Arguments:

`pane` The pane in the dashboard to link to.

Method `print()`**:** Print this object.

Usage:

```
Variant$print(...)
```

Arguments:

`...` Unused.

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`, `Content`, `ContentTask`, `Environment`, `PositConnect`, `Task`, `Vanity`, `VariantSchedule`, `VariantTask`

`VariantSchedule` *VariantSchedule*

Description

VariantSchedule

VariantSchedule

Details

An R6 class that represents a Schedule

Super classes

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

Public fields

`schedule_data` The schedule data.

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()`**:** Initialize this schedule.

Usage:

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

Arguments:

`connect` The `Connect` instance.

`content` The `Content` instance.

`key` The variant key.

`schedule` The schedule data.

Method `GET()`**:** Perform an HTTP GET request of the named API path. Returns an object parsed from the HTTP response.

Usage:

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

Arguments:

`path` API path.

Method `POST()`**:** Perform an HTTP POST request of the named API path. Returns an object parsed from the HTTP response.

Usage:

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

Arguments:

`path` API path.

`body` The HTTP payload.

Method `DELETE()`**:** Perform an HTTP DELETE request of the named API path. Returns the HTTP response object.

Usage:

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

Arguments:

`path` API path.

Method `set_schedule()`**:** Set the schedule for this variant

Usage:

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

Arguments:

`...` Schedule fields.

Method `is_empty()`**:** Return if this variant has a schedule.

Usage:

```
VariantSchedule$is_empty()
```

Method `print()`**:** Print this object.

Usage:

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

Arguments:

`...` Unused.

Method `get_schedule()`**:** Get the schedule data.

Usage:

```
VariantSchedule$get_schedule()
```

Method `get_schedule_remote()`**:** Get and store the (remote) schedule data.

Usage:

```
VariantSchedule$get_schedule_remote()
```

Method `describe_schedule()`**:** Description of the associated 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`, `Content`, `ContentTask`, `Environment`, `PositConnect`, `Task`, `Vanity`, `Variant`, `VariantTask`

VariantTask *VariantTask*

Description

VariantTask

VariantTask

Details

An R6 class that represents a Variant Task

Super classes

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

Public fields

`task` The task.

`data` The variant data.

Methods

Public methods:

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

Method `new()`**:** Initialize this variant task.

Usage:

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

Arguments:

`connect` The `Connect` instance.

`content` The `Content` instance.

`key` The variant key.

`task` The task data.

Method `get_task()`**:** Return the underlying task.

Usage:

```
VariantTask$get_task()
```

Method `add_data()`**:** Set the data.

Usage:

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

Arguments:

`data` The data.

Method `get_data()`**:** Get the data.

Usage:

```
VariantTask$get_data()
```

Method `print()`**:** Print this object.

Usage:

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

Arguments:

`...` Unused.

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`, `Content`, `ContentTask`, `Environment`, `PositConnect`, `Task`, `Vanity`, `Variant`, `VariantSchedule`

`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: content_delete(), content_item(), content_title(), content_update(), create_random_name(), dashboard_url(), delete_thumbnail(), delete_vanity_url(), deploy_repo(), get_associations(), get_bundles(), get_environment(), get_jobs(), get_log(), get_thumbnail(), get_vanity_url(), git, has_thumbnail(), lock_content(), permissions, search_content(), set_integrations(), set_run_as(), set_thumbnail(), set_vanity_url(), swap_vanity_urls(), terminate_jobs()

Index