Package 'rsconnect'

March 9, 2018

Type Package

Title Deployment Interface for R Markdown Documents and Shiny Applications

Version 0.8.8

Author JJ Allaire

Maintainer Jonathan McPherson <jonathan@rstudio.com>

Description Programmatic deployment interface for 'RPubs', 'shinyapps.io', and 'RStudio Connect'. Supported content types include R Markdown documents, Shiny applications, Plumber APIs, plots, and static web content.

Depends R (>= 3.0.0)

Imports digest, PKI, RCurl, RJSONIO, packrat (>= 0.4.8-1), yaml (>= 2.1.5), rstudioapi (>= 0.5)

Suggests knitr, testthat, rmarkdown (>= 1.1), plumber (>= 0.3.2), shiny, sourcetools, xtable

Enhances BiocInstaller

License GPL-2

RoxygenNote 6.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2018-03-09 00:26:52 UTC

R topics documented:

`rsconnect-package` *rsconnect*

Description

Deployment Interface for R Markdown Documents and Shiny Applications

Details

The rsconnect package provides a programmatic deployment interface for RPubs, shinyapps.io, and RStudio Connect. Supported contents types include R Markdown documents, Shiny applications, plots, and static web content.

Managing Applications

Deploy and manage applications with the following functions:

deployApp:	Deploy a Shiny application to a server.
configureApp:	Configure an application currently running on a server.
restartApp:	Restart an application currently running on a server.
terminateApp:	Terminate an application currently running on a server.
deployments:	List deployment records for a given application directory.

More information on application management is available in the applications help page.

Managing Accounts and Users

Manage accounts on the local system.

setAccountInfo:	Register an account.
removeAccount:	Remove an account.
accountInfo:	View information for a given account.

More information on account management is available in the accounts help page.

---

accounts *Account Management Functions*

Description

Functions to enumerate and remove accounts on the local system. Prior to deploying applications you need to register your account on the local system.

Usage

```
accounts(server = NULL)

accountInfo(name, server = NULL)

removeAccount(name, server = NULL)
```

Arguments

server	Name of the server on which the account is registered (optional; see servers)
name	Name of account

Details

You register an account using the setAccountInfo function (for ShinyApps) or connectUser function (for other servers). You can subsequently remove the account using the removeAccount function.

The accounts and accountInfo functions are provided for viewing previously registered accounts.

Value

accounts returns a data frame with the names of all accounts registered on the system and the servers on which they reside. accountInfo returns a list with account details.

---

accountUsage *Show Account Usage*

Description

Show account usage

Usage

```
accountUsage(account = NULL, server = NULL, usageType = "hours",
  from = NULL, until = NULL, interval = NULL)
```

Arguments

account	Account name. If a single account is registered on the system then this parameter can be omitted.
server	Server name. Required only if you use the same account name on multiple servers.
usageType	Use metric to retreive (for example: "hours")
from	Date range starting timestamp (Unix timestamp or relative time delta such as "2d" or "3w").
until	Date range ending timestamp (Unix timestamp or relative time delta such as "2d" or "3w").
interval	Summarization interval. Data points at intervals less then this will be grouped. (Number of seconds or relative time delta e.g. "1h").

Note

This function only works for ShinyApps servers.

addAuthorizedUser *Add authorized user to application*

Description

Add authorized user to application

Usage

```
addAuthorizedUser(email, appDir = getwd(), appName = NULL, account = NULL,
  server = NULL, sendEmail = NULL, emailMessage = NULL)
```

Arguments

email	Email address of user to add.
appDir	Directory containing application. Defaults to current working directory.
appName	Name of application.
account	Account name. If a single account is registered on the system then this parameter can be omitted.
server	Server name. Required only if you use the same account name on multiple servers.
sendEmail	Send an email letting the user know the application has been shared with them.
emailMessage	Optional character vector of length 1 containing a custom message to send in email invitation. Defaults to NULL, which will use default invitation message.

Note

This function works only for ShinyApps servers.

See Also

removeAuthorizedUser and showUsers

addLinter *Add a Linter*

Description

Add a linter, to be used in subsequent calls to lint.

Usage

```
addLinter(name, linter)
```

Arguments

name	The name of the linter, as a string.
linter	A linter.

Examples

```
addLinter("no.capitals", linter(

  ## Identify lines containing capital letters -- either by name or by index
  apply = function(content, ...) {
    grep("[A-Z]", content)
  },

  ## Only use this linter on R files (paths ending with .r or .R)
  takes = function(paths) {
    grep("[rR]$", paths)
  },

  # Use the default message constructor
  message = function(content, lines, ...) {
    makeLinterMessage("Capital letters found on the following lines", content, lines)
  },

  # Give a suggested prescription
  suggest = "Do not use capital letters in these documents."
))
```

---

appDependencies *Detect Application Dependencies*

Description

Recursively detect all package dependencies for an application. This function parses all .R files in the application directory to determine what packages the application depends on; and for each of those packages what other packages they depend on.

Usage

```
appDependencies(appDir = getwd(), appFiles = NULL)
```

Arguments

appDir	Directory containing application. Defaults to current working directory.
appFiles	The files and directories to bundle and deploy (only if upload = TRUE). Can be NULL, in which case all the files in the directory containing the application are bundled. Takes precedence over appFileManifest if both are supplied.

Details

Dependencies are determined by parsing application source code and looking for calls to `library`, `require`, `::`, and `:::`.

Recursive dependencies are detected by examining the `Depends`, `Imports`, and `LinkingTo` fields of the packages immediately dependend on by the application.

Value

Returns a data frame listing the package dependencies detected for the application:

`package`	Name of package
`version`	Version of package

Note

Since the `Suggests` field is not included when determining recursive dependencies of packages, it's possible that not every package required to run your application will be detected.

In this case, you can force a package to be included dependency by inserting call(s) to `require` within your source directory. This code need not actually execute, for example you could create a standalone file named `dependencies.R` with the following code:

```
require(xts)
require(colorspace)
```

This will force the `xts` and `colorspace` packages to be installed along with the rest of your application when it is deployed.

See Also

Using Packages with rsconnect

Examples

```
## Not run:

# dependencies for the app in the current working dir
appDependencies()

# dependencies for an app in another directory
appDependencies("~/projects/shiny/app1")

## End(Not run)
```

applications *List Deployed Applications*

Description

List all applications currently deployed for a given account.

Usage

```
applications(account = NULL, server = NULL)
```

Arguments

account	Account name. If a single account is registered on the system then this parameter can be omitted.
server	Server name. Required only if you use the same account name on multiple servers.

Value

Returns a data frame with the following columns:

name	Name of application
url	URL where application can be accessed
config_url	URL where application can be configured
status	Current status of application. Valid values are pending, deploying, running, terminating, and terminated

Note

To register an account you call the setAccountInfo function.

See Also

deployApp, terminateApp

Examples

```
## Not run:

# list all applications for the default account
applications()

# list all applications for a specific account
applications("myaccount")

# view the list of applications in the data viewer
View(applications())

## End(Not run)
```

authorizedUsers *(Deprecated) List authorized users for an application*

Description

(Deprecated) List authorized users for an application

Usage

```
authorizedUsers(appDir = getwd())
```

Arguments

appDir	Directory containing application. Defaults to current working directory.

configureApp *Configure an Application*

Description

Configure an application running on a remote server.

Usage

```
configureApp(appName, appDir = getwd(), account = NULL, server = NULL,
  redeploy = TRUE, size = NULL, instances = NULL, logLevel = c("normal",
  "quiet", "verbose"))
```

Arguments

appName	Name of application to configure
appDir	Directory containing application. Defaults to current working directory.
account	Account name. If a single account is registered on the system then this parameter can be omitted.
server	Server name. Required only if you use the same account name on multiple servers (see servers)
redeploy	Re-deploy application after its been configured.
size	Configure application instance size
instances	Configure number of application instances
logLevel	One of "quiet", "normal" or "verbose"; indicates how much logging to the console is to be performed. At "quiet" reports no information; at "verbose", a full diagnostic log is captured.

See Also

applications, deployApp

Examples

```
## Not run:

# set instance size for an application
configureApp("myapp", size="xlarge")

## End(Not run)
```

---

connectUser *Connect User Account*

Description

Connect a user account to the package so that it can be used to deploy and manage applications on behalf of the account.

Usage

```
connectUser(account = NULL, server = NULL, quiet = FALSE)
```

Arguments

account	A name for the account to connect. Optional.
server	The server to connect to. Optional if there is only one server registered.
quiet	Whether or not to show messages and prompts while connecting the account.

Details

When this function is invoked, a web browser will be opened to a page on the target server where you will be prompted to enter your credentials. Upon successful authentication, your local installation of **rsconnect** and your server account will be paired, and you'll be able to deploy and manage applications using the package without further prompts for credentials.

deployAPI *Deploy a Plumber API*

Description

Deploys an application consisting of plumber API routes. The given directory must contain a script returning a ‘plumb‘ object or a plumber API definition.

Usage

```
deployAPI(api, ...)
```

Arguments

api	Path to the API project directory. Must contain either ‘entrypoint.R‘ or ‘plumber.R‘
...	Additional arguments to deployApp.

Details

Deploy a plumber API definition by either supplying a directory containing ‘plumber.R‘ (an API definition) or ‘entrypoint.R‘ that returns a ‘plumb‘ object created by ‘plumber::plumb()‘. See the plumber documentation for more information.

deployApp *Deploy an Application*

Description

Deploy a shiny application, an R Markdown document, a plumber API, or HTML content to a server.

Usage

```
deployApp(appDir = getwd(), appFiles = NULL, appFileManifest = NULL,
  appPrimaryDoc = NULL, appSourceDoc = NULL, appName = NULL,
  appTitle = NULL, appId = NULL, contentCategory = NULL, account = NULL,
  server = NULL, upload = TRUE,
  launch.browser = getOption("rsconnect.launch.browser", interactive()),
  logLevel = c("normal", "quiet", "verbose"), lint = TRUE,
  metadata = list(), forceUpdate = getOption("rsconnect.force.update.apps",
  FALSE))
```

Arguments

appDir	Directory containing application. Defaults to current working directory.
appFiles	The files and directories to bundle and deploy (only if upload = TRUE). Can be NULL, in which case all the files in the directory containing the application are bundled. Takes precedence over appFileManifest if both are supplied.
appFileManifest	An alternate way to specify the files to be deployed; a file containing the names of the files, one per line, relative to the appDir.
appPrimaryDoc	If the application contains more than one document, this parameter indicates the primary one, as a path relative to appDir. Can be NULL, in which case the primary document is inferred from the contents being deployed.
appSourceDoc	If the application is composed of static files (e.g HTML), this parameter indicates the source document, if any, as a fully qualified path. Deployment information returned by deployments is associated with the source document.
appName	Name of application (names must be unique within an account). Defaults to the base name of the specified appDir.
appTitle	Free-form descriptive title of application. Optional; if supplied, will often be displayed in favor of the name. When deploying a new application, you may supply only the appTitle to receive an auto-generated appName.
appId	If updating an application, the ID of the application being updated. Optional unless updating an app owned by another user.
contentCategory	Optional; the kind of content being deployed (e.g. "plot", "document", or "application").
account	Account to deploy application to. This parameter is only required for the initial deployment of an application when there are multiple accounts configured on the system (see accounts).
server	Server name. Required only if you use the same account name on multiple servers.
upload	If TRUE (the default) then the application is uploaded from the local system prior to deployment. If FALSE then it is re-deployed using the last version that was uploaded.
launch.browser	If true, the system's default web browser will be launched automatically after the app is started. Defaults to TRUE in interactive sessions only.
logLevel	One of "quiet", "normal" or "verbose"; indicates how much logging to the console is to be performed. At "quiet" reports no information; at "verbose", a full diagnostic log is captured.
lint	Lint the project before initiating deployment, to identify potentially problematic code?
metadata	Additional metadata fields to save with the deployment record. These fields will be returned on subsequent calls to deployments.
forceUpdate	If TRUE, update any previously-deployed app without asking. If FALSE, ask to update. If unset, defaults to the value of getOption("rsconnect.force.update.apps", FALSE).

See Also

applications, terminateApp, and restartApp

Examples

```
## Not run:

# deploy the application in the current working dir
deployApp()

# deploy an application in another directory
deployApp("~/projects/shiny/app1")

# deploy using an alternative application name and title
deployApp("~/projects/shiny/app1", appName = "myapp",
          appTitle = "My Application")

# deploy specifying an explicit account name, then
# redeploy with no arguments (will automatically use
# the previously specified account)
deployApp(account = "jsmith")
deployApp()

# deploy but don't launch a browser when completed
deployApp(launch.browser = FALSE)

## End(Not run)
```

---

deployDoc *Deploy a Document*

Description

Deploys an application consisting of a single R Markdown document or other single file (such as an HTML or PDF document).

Usage

```
deployDoc(doc, ...)
```

Arguments

doc	Path to the document to deploy.
...	Additional arguments to deployApp. Do not supply appDir, appFiles, or appPrimaryDoc; these three parameters are automatically generated by deployDoc from the document.

Details

When deploying an R Markdown document, any files which are required to render and display the file must be deployed.

This method discovers these additional files using `find_external_resources` from **rmarkdown**.

If you find that the document is missing dependencies, either specify the dependencies explicitly in the document (the documentation for `find_external_resources` explains how to do this), or call `deployApp` directly and specify your own file list in the `appFiles` parameter.

`deployments` *List Application Deployments*

Description

List deployment records for a given application.

Usage

```
deployments(appPath, nameFilter = NULL, accountFilter = NULL,
  serverFilter = NULL, excludeOrphaned = TRUE)
```

Arguments

`appPath`	The path to the content that was deployed, either a directory or an individual document.
`nameFilter`	Return only deployments matching the given name (optional)
`accountFilter`	Return only deployments matching the given account (optional)
`serverFilter`	Return only deployments matching the given server (optional)
`excludeOrphaned`	If `TRUE` (the default), return only deployments made by a currently registered account. Deployments made from accounts that are no longer registered (via e.g.`removeAccount`) will not be returned.

Value

Returns a data frame with at least following columns:

`name`	Name of deployed application
`account`	Account owning deployed application
`bundleId`	Identifier of deployed application's bundle
`url`	URL of deployed application
`when`	When the application was deployed (in seconds since the epoch)

If additional metadata has been saved with the deployment record using the `metadata` argument to `deployApp`, the frame will include additional columns.

See Also

applications to get a list of deployments from the server, and deployApp to create a new deployment.

Examples

```
## Not run:

# Return all deployments of the ~/r/myapp directory made with the 'abc'
# account
deployments("~/r/myapp", accountFilter="abc")

## End(Not run)
```

---

deploySite *Deploy a Website*

Description

Deploy an R Markdown website to a server.

Usage

```
deploySite(siteDir = getwd(), siteName = NULL, account = NULL,
  server = NULL, render = c("none", "local", "server"),
  launch.browser = getOption("rsconnect.launch.browser", interactive()),
  logLevel = c("normal", "quiet", "verbose"), lint = FALSE,
  metadata = list())
```

Arguments

siteDir	Directory containing website. Defaults to current working directory.
siteName	Name for the site (names must be unique within an account). Defaults to the base name of the specified siteDir, (or to a name provided by a custom site generation function).
account	Account to deploy application to. This parameter is only required for the initial deployment of an application when there are multiple accounts configured on the system (see accounts).
server	Server name. Required only if you use the same account name on multiple servers.
render	Rendering behavior for site: "none" to upload a static version of the current contents of the site directory; "local" to render the site locally then upload it; "server" to render the site on the server. Note that for "none" and "local" R scripts (.R) and markdown documents (.Rmd and .md) will not be uploaded to the server.

launch.browser	If true, the system's default web browser will be launched automatically after the app is started. Defaults to TRUE in interactive sessions only.
logLevel	One of "quiet", "normal" or "verbose"; indicates how much logging to the console is to be performed. At "quiet" reports no information; at "verbose", a full diagnostic log is captured.
lint	Lint the project before initiating deployment, to identify potentially problematic code?
metadata	Additional metadata fields to save with the deployment record. These fields will be returned on subsequent calls to deployments.

---

deployTFModel *Deploy a TensorFlow saved model*

Description

Deploys a directory containing a Tensorflow saved model file. A saved model directory might look like this:

```
./1/
./1/saved_model.pb or ./1/saved_model.pbtxt
./1/variables/
./1/variables/variables.data-00000-of-00001
./1/variables/variables.index
```

For information on creating saved models, see the Keras method export_savedmodel.keras.engine.training.Model or the TensorFlow method export_savedmodel. If using the TensorFlow package for R, the official TensorFlow guide for saving and restoring models may be useful: https://www.tensorflow.org/programmers_guide/saved_model#overview_of_saving_and_restoring_models

Usage

```
deployTFModel(modelDir, ...)
```

Arguments

modelDir	Path to the saved model directory. MUST contain *saved_model.pb* or *saved_model.pbtxt*
...	Additional arguments to deployApp.

Details

Deploy a single Tensorflow saved model as a bundle. Should be passed a directory that contains the *saved_model.pb* or *saved_model.pbtxt* file, as well as any variables and assets necessary to load the model.

generateAppName *Generate Application Name*

Description

Generate a short name (identifier) for an application given an application title.

Usage

```
generateAppName(appTitle, appPath = NULL, account = NULL, unique = TRUE)
```

Arguments

appTitle	A descriptive title for the application.
appPath	The path to the application's content, either a directory or an individual document. Optional.
account	The account where the application will be deployed. Optional.
unique	Whether to try to generate a unique name.

Details

This function modifies the title until it forms a suitable application name. Suitable application names are 3 - 64 characters long and contain only alphanumeric characters.

The function is intended to be used to find a name for a new application. If appPath and account are both specified, then the returned name will also be unique among locally known deployments of the directory (note that it is not guaranteed to be unique on the server). This behavior can be disabled by setting unique = FALSE.

Value

Returns a valid short name for the application.

Examples

```
# Generate a short name for a sample application
generateAppName("My Father's Country", "~/fathers-country", "myacct")
```

lint *Lint a Project*

Description

Takes the set of active linters (see addLinter), and applies them to all files within a project.

Usage

```
lint(project, files = NULL, appPrimaryDoc = NULL)
```

Arguments

project	Path to a project directory.
files	Specific files to lint. Can be NULL, in which case all the files in the directory will be linted.
appPrimaryDoc	The primary file in the project directory. Can be NULL, in which case it's inferred (if possible) from the directory contents.

linter *Create a Linter*

Description

Generate a linter, which can identify errors or problematic regions in a project.

Usage

```
linter(apply, takes, message, suggestion)
```

Arguments

apply	Function that, given the content of a file, returns the indices at which problems were found.
takes	Function that, given a set of paths, returns the subset of paths that this linter uses.
message	Function that, given content and lines, returns an informative message for the user. Typically generated with makeLinterMessage.
suggestion	String giving a prescribed fix for the linted problem.

Examples

```
addLinter("no.capitals", linter(

  ## Identify lines containing capital letters -- either by name or by index
  apply = function(content, ...) {
    grep("[A-Z]", content)
  },

  ## Only use this linter on R files (paths ending with .r or .R)
  takes = function(paths) {
    grep("[rR]$", paths)
  },

  # Use the default message constructor
  message = function(content, lines, ...) {
    makeLinterMessage("Capital letters found on the following lines", content, lines)
  },

  # Give a suggested prescription
  suggest = "Do not use capital letters in these documents."
))
```

`listBundleFiles` *List Files to be Bundled*

Description

Given a directory containing an application, returns the names of the files to be bundled in the application.

Usage

```
listBundleFiles(appDir)
```

Arguments

`appDir`	Directory containing the application.

Details

This function computes results similar to a recursive directory listing from `list.files`, with the following constraints:

1. If the total size of the files exceeds the maximum bundle size, no more files are listed. The maximum bundle size is controlled by the `rsconnect.max.bundle.size` option.
2. If the total size number of files exceeds the maximum number to be bundled, no more files are listed. The maximum number of files in the bundle is controlled by the `rsconnect.max.bundle.files` option.
3. Certain files and folders that don't need to be bundled, such as those containing internal version control and RStudio state, are excluded.

Value

Returns a list containing the following elements:

`contents`	A list of the files to be bundled
`totalSize`	The total size of the files

`makeLinterMessage` *Construct a Linter Message*

Description

Pretty-prints a linter message. Primarily used as a helper for constructing linter messages with `linter`.

Usage

```
makeLinterMessage(header, content, lines)
```

Arguments

`header`	A header message describing the linter.
`content`	The content of the file that was linted.
`lines`	The line numbers from `content` that contain lint.

`removeAuthorizedUser` *Remove authorized user from an application*

Description

Remove authorized user from an application

Usage

```
removeAuthorizedUser(user, appDir = getwd(), appName = NULL,
  account = NULL, server = NULL)
```

Arguments

`user`	The user to remove. Can be id or email address.
`appDir`	Directory containing application. Defaults to current working directory.
`appName`	Name of application.
`account`	Account name. If a single account is registered on the system then this parameter can be omitted.
`server`	Server name. Required only if you use the same account name on multiple servers.

Note

This function works only for ShinyApps servers.

See Also

addAuthorizedUser and showUsers

---

restartApp *Restart an Application*

Description

Restart an application currently running on a remote server.

Usage

```
restartApp(appName, account = NULL, server = NULL, quiet = FALSE)
```

Arguments

appName	Name of application to restart
account	Account name. If a single account is registered on the system then this parameter can be omitted.
server	Server name. Required only if you use the same account name on multiple servers (see servers)
quiet	Request that no status information be printed to the console during the operation.

See Also

applications, deployApp, and terminateApp

Examples

```
## Not run:

# restart an application
restartApp("myapp")

## End(Not run)
```

rpubsUpload *Upload a file to RPubs*

Description

This function publishes a file to rpubs.com. If the upload succeeds a list that includes an `id` and `continueUrl` is returned. A browser should be opened to the `continueUrl` to complete publishing of the document. If an error occurs then a diagnostic message is returned in the `error` element of the list.

Usage

```
rpubsUpload(title, contentFile, originalDoc, id = NULL, properties = list())
```

Arguments

`title`	The title of the document.
`contentFile`	The path to the content file to upload.
`originalDoc`	The document that was rendered to produce the `contentFile`. May be `NULL` if the document is not known.
`id`	If this upload is an update of an existing document then the id parameter should specify the document id to update. Note that the id is provided as an element of the list returned by successful calls to `rpubsUpload`.
`properties`	A named list containing additional document properties (RPubs doesn't currently expect any additional properties, this parameter is reserved for future use).

Value

A named list. If the upload was successful then the list contains a `id` element that can be used to subsequently update the document as well as a `continueUrl` element that provides a URL that a browser should be opened to in order to complete publishing of the document. If the upload fails then the list contains an `error` element which contains an explanation of the error that occurred.

Examples

```
## Not run:
# upload a document
result <- rpubsUpload("My document title", "Document.html")
if (!is.null(result$continueUrl))
   browseURL(result$continueUrl)
else
   stop(result$error)

# update the same document with a new title
updateResult <- rpubsUpload("My updated title", "Document.html",
                            id = result$id)

## End(Not run)
```

rsconnectOptions *Package Options*

Description

The **rsconnect** package supports several options that control the method used for http communications, the printing of diagnostic information for http requests, and the launching of an external browser after deployment.

Details

Supported global options include:

`rsconnect.ca.bundle` Path to a custom bundle of Certificate Authority root certificates to use when connecting to servers via SSL. This option can also be specied in the environment variable `RSCONNECT_CA_BUNDLE`. Leave undefined to use your system's default certificate store.

`rsconnect.check.certificate` Whether to check the SSL certificate when connecting to a remote host; defaults to `TRUE`. Setting to `FALSE` is insecure, but will allow you to connect to hosts using invalid certificates as a last resort.

`rsconnect.http` Http implementation used for connections to the back-end service:

`rcurl`	Secure https using the `RCurl` package
`curl`	Secure https using the curl system utility
`internal`	Insecure http using raw sockets

If no option is specified then `rcurl` is used by default.

`rsconnect.http.trace` When `TRUE`, trace http calls (prints the method, path, and total milliseconds for each http request)

`rsconnect.http.trace.json` When `TRUE`, trace JSON content (shows JSON payloads sent to and received from the server))

`rsconnect.http.verbose` When `TRUE`, print verbose output for http connections (useful only for debugging SSL certificate or http connection problems)

`rsconnect.error.trace` When `TRUE`, print detailed stack traces for errors occurring during deployment.

`rsconnect.launch.browser` When `TRUE`, automatically launch a browser to view applications after they are deployed

`rsconnect.locale.cache` When `FALSE`, disable the detected locale cache (Windows only).

`rsconnect.locale` Override the detected locale.

`rsconnect.max.bundle.size` The maximum size, in bytes, for deployed content. If not set, defaults to 3 GB.

`rsconnect.max.bundle.files` The maximum number of files to deploy. If not set, defaults to 10,000.

`rsconnect.force.update.apps` When `TRUE`, bypasses the prompt to confirm whether you wish to update previously-deployed content

Examples

```
## Not run:

# use curl for http connections
options(rsconnect.http = "curl")

# trace http requests
options(rsconnect.http.trace = TRUE)

# print verbose output for http requests
options(rsconnect.http.verbose = TRUE)

# print JSON content
options(rsconnect.http.trace.json = TRUE)

# don't automatically launch a browser after deployment
options(rsconnect.launch.browser = FALSE)

## End(Not run)
```

---

rsconnectPackages *Using Packages with rsconnect*

Description

Deployed applications can depend on any package available on CRAN as well as any package hosted in a public GitHub repository.

When an application is deployed it's source code is scanned for dependencies using the `appDependencies` function. The list of dependencies is sent to the server along with the application source code and these dependencies are then installed alongside the application.

Note that the `Suggests` dependencies of packages are not automatically included in the list of dependent packages. See the *Note* section of the documentation of the appDependencies function for details on how to force packages to be included in the dependency list.

CRAN Packages

When sastisfying CRAN package dependencies, the server will build the exact versions of packages that were installed on the system from which the application is deployed.

If a locally installed package was not obtained from CRAN (e.g. was installed from R-Forge) and as a result doesn't have a version that matches a version previously published to CRAN then an error will occur. It's therefore important that you run against packages installed directly from CRAN in your local configuration.

GitHub Packages

It's also possible to depend on packages hosted in public GitHub repositories, so long as they are installed via the `install_github` function from the **devtools** package.

This works because `install_github` records the exact Github commit that was installed locally, making it possible to download and install the same source code on the deployment server.

Note that in order for this to work correctly you need to install the very latest version of `devtools` from Github. You can do this as follows:

```
library(devtools)
install_github("devtools", "hadley")
```

See Also

`appDependencies`

`rsconnectProxies` *HTTP Proxy Configuration*

Description

If your system is behind an HTTP proxy then additional configuration may be required to connect to the ShinyApp service. The required configuration varies depending on what type of HTTP connection you are making to the server.

The default HTTP connection type is `rcurl` however addition connection types `curl` and `internal` are also supported. The HTTP connection type is configured using the `rsconnect.http` global option.

HTTP Proxy Environment Variable

The most straightforward way to specify a proxy for `rcurl` and `curl` connections is to set the `http_proxy` environment variable. For example, you could add the following code to your `.Rprofile`:

```
Sys.setenv(http_proxy = "http://proxy.example.com")
```

Proxy settings can include a host-name, port, and username/password if necessary. The following are all valid values for the `http_proxy` environment variable:

```
http://proxy.example.com/
http://proxy.example.com:1080/
http://username:password@proxy.example.com:1080/
```

Setting RCurl Proxy Options

The default HTTP connection type is `rcurl`. If you need more configurability than afforded by the `http_proxy` environment variable you can specify RCurl proxy options explicity using `RCurlOptions`. For example, you could add the following code to your `.Rprofile`:

```
options(RCurlOptions = list(proxy = "http://proxy.example.com")
```

You can set any underling curl option using this mechanism. To do this you translate curl options to lowercase and remove the `CURL_` prefix (for example, `CURLOPT_PROXYPORT` becomes `proxyport`).

Using Internet Explorer Proxy Settings

If you are running on Windows and have difficulty configuring proxy settings for `rcurl` or `curl` connections, it's possible to re-use your Internet Explorer proxy settings for connections to the server. To do this you set the http connection type to `internal` as follows:

```
options(rsconnect.http = "internal")
```

The `internal` connection type uses an insecure (non-encrypted) http connection to the server. If you require an encrypted https connection it's recommended that you use an `rcurl` or `curl` connection.

---

servers *Server Management Functions*

Description

Functions to manage the list of known servers to which **rsconnect** can deploy and manage applications.

Usage

```
servers(local = FALSE)

discoverServers(quiet = FALSE)

addConnectServer(url, name = NULL, certificate = NULL, quiet = FALSE)

addServer(url, name = NULL, certificate = NULL, quiet = FALSE)

removeServer(name)

serverInfo(name)

addServerCertificate(name, certificate, quiet = FALSE)
```

Arguments

local	Return only local servers (i.e. not shinyapps.io)
quiet	Suppress output and prompts where possible.
url	Server's URL. Should look like http://servername/ or http://servername:port/.
name	Optional nickname for the server. If none is given, the nickname is inferred from the server's hostname.
certificate	Optional; a path a certificate file to be used when making SSL connections to the server. The file's contents are copied and stored by the **rsconnect** package. Can also be a character vector containing the certificate's contents.

Details

Register a server with addServer or discoverServers (the latter is useful only if your administrator has configured server autodiscovery). Once a server is registered, you can connect to an account on the server using connectUser.

The servers and serverInfo functions are provided for viewing previously registered servers.

There is always at least one server registered (the shinyapps.io server)

Value

servers returns a data frame with registered server names and URLs. serverInfo returns a list with details for a particular server.

Examples

```
## Not run:

# register a local server
addServer("http://myrsconnect/", "myserver")

# list servers
servers(local = TRUE)

# connect to an account on the server
connectUser(server = "myserver")

## End(Not run)
```

---

setAccountInfo *Set ShinyApps Account Info*

Description

Configure a ShinyApps account for publishing from this system.

Usage

```
setAccountInfo(name, token, secret)
```

Arguments

name	Name of account to save or remove
token	User token for the account
secret	User secret for the account

Examples

```
## Not run:

# register an account
setAccountInfo("user", "token", "secret")

# remove the same account
removeAccount("user")

## End(Not run)
```

---

setProperty *Set Application property*

Description

Set a property on currently deployed ShinyApps application.

Usage

```
setProperty(propertyName, propertyValue, appPath = getwd(), appName = NULL,
  account = NULL, force = FALSE)
```

Arguments

propertyName	Name of property to set
propertyValue	Nalue to set property to
appPath	Directory or file that was deployed. Defaults to current working directory.
appName	Name of application
account	Account name. If a single account is registered on the system then this parameter can be omitted.
force	Forcibly set the property

Note

This function only works for ShinyApps servers.

Examples

```
## Not run:

# set instance size for an application
setProperty("application.instances.count", 1)

# disable application package cache
setProperty("application.package.cache", FALSE)


## End(Not run)
```

`showInvited` *List invited users for an application*

Description

List invited users for an application

Usage

```
showInvited(appDir = getwd(), appName = NULL, account = NULL,
  server = NULL)
```

Arguments

`appDir`	Directory containing application. Defaults to current working directory.
`appName`	Name of application.
`account`	Account name. If a single account is registered on the system then this parameter can be omitted.
`server`	Server name. Required only if you use the same account name on multiple servers.

Note

This function works only for ShinyApps servers.

See Also

`addAuthorizedUser` and `showUsers`

showLogs *Show Application Logs*

Description

Show the logs for a deployed ShinyApps application.

Usage

```
showLogs(appPath = getwd(), appFile = NULL, appName = NULL,
  account = NULL, entries = 50, streaming = FALSE)
```

Arguments

appPath	The path to the directory or file that was deployed.
appFile	The path to the R source file that contains the application (for single file applications).
appName	The name of the application to show logs for. May be omitted if only one application deployment was made from appPath.
account	The account under which the application was deployed. May be omitted if only one account is registered on the system.
entries	The number of log entries to show. Defaults to 50 entries.
streaming	Whether to stream the logs. If TRUE, then the function does not return; instead, log entries are written to the console as they are made, until R is interrupted. Defaults to FALSE.

Note

This function works only for ShinyApps servers.

showMetrics *Show Application Metrics*

Description

Show application metrics of a currently deployed application

Usage

```
showMetrics(metricSeries, metricNames, appDir = getwd(), appName = NULL,
  account = NULL, server = NULL, from = NULL, until = NULL,
  interval = NULL)
```

Arguments

metricSeries	Metric series to query e.g. "container.cpu"
metricNames	Metric names in the series to query e.g. c("cpu.user", "cpu.system")
appDir	Directory containing application. Defaults to current working directory.
appName	Name of application
account	Account name. If a single account is registered on the system then this parameter can be omitted.
server	Server name. Required only if you use the same account name on multiple servers.
from	Date range starting timestamp (Unix timestamp or relative time delta such as "2d" or "3w").
until	Date range ending timestamp (Unix timestamp or relative time delta such as "2d" or "3w").
interval	Summarization interval. Data points at intervals less then this will be grouped. (Number of seconds or relative time delta e.g. "1h").

Note

This function only works for ShinyApps servers.

---

showProperties *Show Application property*

Description

Show propreties of an application deployed to ShinyApps.

Usage

```
showProperties(appPath = getwd(), appName = NULL, account = NULL)
```

Arguments

appPath	Directory or file that was deployed. Defaults to current working directory.
appName	Name of application
account	Account name. If a single account is registered on the system then this parameter can be omitted.

Note

This function works only for ShinyApps servers.

showUsage *Show Application Usage*

Description

Show application usage of a currently deployed application

Usage

```
showUsage(appDir = getwd(), appName = NULL, account = NULL,
  server = NULL, usageType = "hours", from = NULL, until = NULL,
  interval = NULL)
```

Arguments

appDir	Directory containing application. Defaults to current working directory.
appName	Name of application
account	Account name. If a single account is registered on the system then this parameter can be omitted.
server	Server name. Required only if you use the same account name on multiple servers.
usageType	Use metric to retreive (for example: "hours")
from	Date range starting timestamp (Unix timestamp or relative time delta such as "2d" or "3w").
until	Date range ending timestamp (Unix timestamp or relative time delta such as "2d" or "3w").
interval	Summarization interval. Data points at intervals less then this will be grouped. (Number of seconds or relative time delta e.g. "1h").

Note

This function only works for ShinyApps servers.

showUsers *List authorized users for an application*

Description

List authorized users for an application

Usage

```
showUsers(appDir = getwd(), appName = NULL, account = NULL,
  server = NULL)
```

Arguments

appDir	Directory containing application. Defaults to current working directory.
appName	Name of application.
account	Account name. If a single account is registered on the system then this parameter can be omitted.
server	Server name. Required only if you use the same account name on multiple servers.

Note

This function works only for ShinyApps servers.

See Also

addAuthorizedUser and showInvited

---

taskLog *Show task log*

Description

Writes the task log for the given task

Usage

```
taskLog(taskId, account = NULL, server = NULL, output = NULL)
```

Arguments

taskId	Task Id
account	Account name. If a single account is registered on the system then this parameter can be omitted.
server	Server name. Required only if you use the same account name on multiple servers (see servers)
output	Where to write output. Valid values are NULL or stderr

See Also

tasks

Examples

```
## Not run:

# write task log to stdout
taskLog(12345)

# write task log to stderr
taskLog(12345, output="stderr")


## End(Not run)
```

`tasks` *List Tasks*

Description

List Tasks

Usage

```
tasks(account = NULL, server = NULL)
```

Arguments

`account`	Account name. If a single account is registered on the system then this parameter can be omitted.
`server`	Server name. Required only if you use the same account name on multiple servers (see `servers`)

Value

Returns a data frame with the following columns:

`id`	Task id
`action`	Task action
`status`	Current task status
`created_time`	Task creation time
`finished_time`	Task finished time

See Also

`taskLog`

Examples

```
## Not run:
```

```
# list tasks for the default account
tasks()


## End(Not run)
```

---

terminateApp *Terminate an Application*

Description

Terminate and archive a currently deployed ShinyApps application.

Usage

```
terminateApp(appName, account = NULL, server = NULL, quiet = FALSE)
```

Arguments

appName	Name of application to terminate
account	Account name. If a single account is registered on the system then this parameter can be omitted.
server	Server name. Required only if you use the same account name on multiple servers (see servers)
quiet	Request that no status information be printed to the console during the termination.

Note

This function only works for ShinyApps servers.

See Also

applications, deployApp, and restartApp

Examples

```
## Not run:

# terminate an application
terminateApp("myapp")

## End(Not run)
```

unsetProperty *Unset Application property*

Description

Unset a property on currently deployed ShinyApps application (restoring to its default value)

Usage

```
unsetProperty(propertyName, appPath = getwd(), appName = NULL,
  account = NULL, force = FALSE)
```

Arguments

propertyName	Name of property to unset
appPath	Directory or file that was deployed. Defaults to current working directory.
appName	Name of application
account	Account name. If a single account is registered on the system then this parameter can be omitted.
force	Forcibly unset the property

Note

This function only works for ShinyApps servers.

Examples

```
## Not run:

# unset application package cache property to revert to default
unsetProperty("application.package.cache")


## End(Not run)
```

Index