Package ‘aws.s3’

April 7, 2020

Type Package

Title 'AWS S3' Client Package

Version 0.3.21

Maintainer Simon Urbanek <`simon.urbanek@R-project.org`>

Description A simple client package for the Amazon Web Services ('AWS') Simple Storage Service ('S3') 'REST' 'API' <https://aws.amazon.com/s3/>.

License GPL (>= 2)

URL `https://github.com/cloudyr/aws.s3`

BugReports `https://github.com/cloudyr/aws.s3/issues`

Encoding UTF-8

Imports utils, tools, curl, httr, xml2 (> 1.0.0), base64enc, digest, aws.signature (>= 0.3.7)

Suggests testthat, datasets

RoxygenNote 7.0.2

NeedsCompilation no

Author Thomas J. Leeper [aut] (<https://orcid.org/0000-0003-4097-6326>),
Boettiger Carl [ctb],
Andrew Martin [ctb],
Mark Thompson [ctb],
Tyler Hunt [ctb],
Steven Akins [ctb],
Bao Nguyen [ctb],
Thierry Onkelinx [ctb],
Andrii Degtiarov [ctb],
Dhruv Aggarwal [ctb],
Alyssa Columbus [ctb],
Simon Urbanek [cre, ctb]

Repository CRAN

Date/Publication 2020-04-07 21:10:02 UTC

R topics documented:

`aws.s3-package` *aws.s3-package*

Description

AWS S3 Client Package

Details

A simple client package for the Amazon Web Services (AWS) Simple Storage Service (S3) REST API.

Author(s)

Thomas J. Leeper <thosjleeper@gmail.com>

`bucketlist` *List Buckets*

Description

List buckets as a data frame

Usage

```
bucketlist(add_region = FALSE, ...)

bucket_list_df(add_region = FALSE, ...)
```

Arguments

`add_region`	A logical (by default `FALSE`) indicating whether to add "Region" column to the output data frame. This simply induces a loop over `get_location` for each bucket.
`...`	Additional arguments passed to `s3HTTP`.

Details

`bucketlist` performs a GET operation on the base s3 endpoint and returns a list of all buckets owned by the authenticated sender of the request. If authentication is successful, this function provides a list of buckets available to the authenticated user. In this way, it can serve as a "hello world!" function, to confirm that one's authentication credentials are working correctly.

`bucket_list_df` and `bucketlist` are identical.

Value

A data frame of buckets. Can be empty (0 rows, 0 columns) if there are no buckets, otherwise contains typically at least columns `Bucket` and `CreationDate`.

References

API Documentation

See Also

`get_bucket`, `get_object`

bucket_exists *Bucket exists?*

Description

Check whether a bucket exists and is accessible with the current authentication keys.

Usage

```
bucket_exists(bucket, ...)
```

Arguments

`bucket`	Character string with the name of the bucket, or an object of class "s3_bucket".
`...`	Additional arguments passed to `s3HTTP`.

Value

`TRUE` if bucket exists and is accessible, else `FALSE`.

References

API Documentation

See Also

`bucketlist`, `get_bucket`, `object_exists`

copy_object *Copy Objects*

Description

Copy objects between S3 buckets

Usage

```
copy_object(
  from_object,
  to_object = from_object,
  from_bucket,
  to_bucket,
  headers = list(),
  ...
)

copy_bucket(from_bucket, to_bucket, ...)
```

Arguments

`from_object`	A character string containing the name the object you want to copy.
`to_object`	A character string containing the name the object should have in the new bucket.
`from_bucket`	A character string containing the name of the bucket you want to copy from.
`to_bucket`	A character string containing the name of the bucket you want to copy into.
`headers`	List of request headers for the REST call.
`...`	Additional arguments passed to `s3HTTP`.

Details

`copy_object` copies an object from one bucket to another without bringing it into local memory. For `copy_bucket`, all objects from one bucket are copied to another (limit 1000 objects). The same keys are used in the old bucket as in the new bucket.

Value

Something...

References

API Documentation

`delete_bucket` *Delete Bucket*

Description

Deletes an S3 bucket.

Usage

```
delete_bucket(bucket, ...)
```

Arguments

`bucket`	Character string with the name of the bucket, or an object of class “s3_bucket”.
`...`	Additional arguments passed to `s3HTTP`.

Value

`TRUE` if successful, `FALSE` otherwise.

References

API Documentation

`delete_object` *Delete object*

Description

Deletes one or more objects from an S3 bucket.

Usage

```
delete_object(object, bucket, quiet = TRUE, ...)
```

Arguments

`object`	Character string with the object key, or an object of class "s3_object". In most cases, if `object` is specified as the latter, `bucket` can be omitted because the bucket name will be extracted from "Bucket" slot in `object`.
`bucket`	Character string with the name of the bucket, or an object of class "s3_bucket".
`quiet`	A logical indicating whether (when `object` is a list of multiple objects), to run the operation in "quiet" mode. Ignored otherwise. See API documentation for details.
`...`	Additional arguments passed to `s3HTTP`.

Details

`object` can be a single object key, an object of class "s3_object", or a list of either.

Value

`TRUE` if successful, otherwise an object of class aws_error details if not.

References

API Documentation

See Also

`put_object`

`delete_website` *Bucket Website configuration*

Description

Get/Put/Delete the website configuration for a bucket.

Usage

```
delete_website(bucket, ...)

put_website(bucket, request_body, ...)

get_website(bucket, ...)
```

Arguments

`bucket`	Character string with the name of the bucket, or an object of class "s3_bucket".
`...`	Additional arguments passed to `s3HTTP`.
`request_body`	A character string containing an XML request body, as defined in the specification in the API Documentation.

Value

For `put_website` and `get_website`, a list containing the website configuration, if one has been set. For `delete_website`: `TRUE` if successful, `FALSE` otherwise. An `aws_error` object may be returned if the request failed.

References

API Documentation: PUT website API Documentation: GET website API Documentation: DELETE website

`getobject` *Deprecated*

Description

These functions are deprecated.

Usage

```
getobject(...)

saveobject(...)

headobject(...)

copyobject(...)

copybucket(...)

putbucket(...)

putobject(...)

deleteobject(...)

getbucket(...)

deletebucket(...)

bucketexists(...)
```

Arguments

`...`	Arguments passed to updated versions of each function.

`get_acceleration` *Bucket Acceleration*

Description

Get/Put acceleration settings or retrieve acceleration status of a bucket.

Usage

```
get_acceleration(bucket, ...)

put_acceleration(bucket, status = c("Enabled", "Suspended"), ...)
```

Arguments

`bucket`	Character string with the name of the bucket, or an object of class “s3_bucket”.
`...`	Additional arguments passed to `s3HTTP`.
`status`	Character string specifying whether acceleration should be “Enabled” or “Suspended”.

Details

Transfer acceleration is a AWS feature that enables potentially faster file transfers to and from S3, particularly when making cross-border transfers (such as from a European client location to the ‘us-east-1’ S3 region). Acceleration must be enabled before it can be used. Once enabled, `accelerate = TRUE` can be passed to any aws.s3 function via `s3HTTP`. `get_acceleration` returns the acceleration status of a bucket; `put_acceleration` enables or suspends acceleration.

Value

For `get_acceleration`: If acceleration has never been enabled or suspend, the value is `NULL`. Otherwise, the status is returned (either “Enabled” or “Suspended”). For `put_acceleration`: If acceleration has never been enabled or suspend, the value is `NULL`.

References

API Documentation: PUT Bucket accelerate API Documentation: GET Bucket accelerate

Examples

```
## Not run:
b <- bucketlist()
get_acceleration(b[[1]])
put_acceleration(b[[1]], "Enabled")
get_acceleration(b[[1]])
put_acceleration(b[[1]], "Suspended")

## End(Not run)
```

`get_acl` *Get or put bucket/object ACLs*

Description

Access Control Lists (ACLs) control access to buckets and objects. These functions retrieve and modify ACLs for either objects or buckets.

Usage

```
get_acl(object, bucket, ...)

put_acl(object, bucket, acl = NULL, headers = list(), body = NULL, ...)
```

Arguments

`object`	Character string with the object key, or an object of class “s3_object”. In most cases, if `object` is specified as the latter, `bucket` can be omitted because the bucket name will be extracted from “Bucket” slot in `object`.
`bucket`	Character string with the name of the bucket, or an object of class “s3_bucket”.

`...`	Additional arguments passed to `s3HTTP`.
`acl`	A character string indicating a "canned" access control list. By default all bucket contents and objects therein are given the ACL "private". This can later be viewed using `get_acl` and modified using `put_acl`.
`headers`	List of request headers for the REST call
`body`	A character string containing an XML-formatted ACL.

Details

`get_acl` retrieves an XML-formatted ACL for either an object (if specified) or a bucket (if specified).

Value

For `get_acl` a character string containing an XML-formatted ACL. For `put_acl`: if successful, `TRUE`.

References

API Reference: GET Object ACL API Reference: PUT Object ACL

`get_bucket` *List bucket contents*

Description

List the contents of an S3 bucket as either a list or data frame

Usage

```
get_bucket(
  bucket,
  prefix = NULL,
  delimiter = NULL,
  max = NULL,
  marker = NULL,
  parse_response = TRUE,
  ...
)

get_bucket_df(
  bucket,
  prefix = NULL,
  delimiter = NULL,
  max = NULL,
  marker = NULL,
  ...
)
```

Arguments

`bucket`	Character string with the name of the bucket, or an object of class "s3_bucket".
`prefix`	Character string that limits the response to keys that begin with the specified prefix
`delimiter`	Character string used to group keys. Read the AWS doc for more detail.
`max`	Integer indicating the maximum number of keys to return. The function will recursively access the bucket in case `max > 1000`. Use `max = Inf` to retrieve all objects.
`marker`	Character string that specifies the key to start with when listing objects in a bucket. Amazon S3 returns object keys in alphabetical order, starting with key after the marker in order.
`parse_response`	logical, should we attempt to parse the response?
`...`	Additional arguments passed to `s3HTTP`.

Details

From the AWS doc: "This implementation of the GET operation returns some or all (up to 1000) of the objects in a bucket. You can use the request parameters as selection criteria to return a subset of the objects in a bucket." The `max` and `marker` arguments can be used to retrieve additional pages of results. Values from a call are store as attributes

Value

`get_bucket` returns a list of objects in the bucket (with class "s3_bucket"), while `get_bucket_df` returns a data frame (the only difference is the application of the `as.data.frame()` method to the list of bucket contents. If `max` is greater than 1000, multiple API requests are executed and the attributes attached to the response object reflect only the final request.

References

API Documentation

See Also

`bucketlist`, `get_object`

Examples

```
## Not run:
  # basic usage
  b <- bucketlist()
  get_bucket(b[1,1])
  get_bucket_df(b[1,1])

  # bucket names with dots
  ## this (default) should work:
  get_bucket("this.bucket.has.dots", url_style = "path")
  ## this probably wont:
```

```
  #get_bucket("this.bucket.has.dots", url_style = "virtual")

## End(Not run)
```

get_bucketname *Utility Functions*

Description

Some utility functions for working with S3 objects and buckets

Usage

```
get_bucketname(x, ...)

## S3 method for class 'character'
get_bucketname(x, ...)

## S3 method for class 's3_bucket'
get_bucketname(x, ...)

## S3 method for class 's3_object'
get_bucketname(x, ...)

get_objectkey(x, ...)

## S3 method for class 'character'
get_objectkey(x, ...)

## S3 method for class 's3_object'
get_objectkey(x, ...)
```

Arguments

x	S3 object, s3:// URL or a string
...	Ignored.

Value

`get_bucketname` returns a character string with the name of the bucket.

`get_objectkey` returns a character string with S3 key which is the part excluding bucket name and leading slashes

`get_bucket_policy` *Bucket policies*

Description

Get/Put/Delete the bucket access policy for a bucket.

Usage

```
get_bucket_policy(bucket, parse_response = TRUE, ...)

put_bucket_policy(bucket, policy, ...)

delete_bucket_policy(bucket, ...)
```

Arguments

`bucket`	Character string with the name of the bucket, or an object of class “s3_bucket”.
`parse_response`	A logical indicating whether to return the response as is, or parse and return as a list. Default is `FALSE`.
`...`	Additional arguments passed to `s3HTTP`.
`policy`	A character string containing a bucket policy.

Details

Bucket policies regulate who has what access to a bucket and its contents. The `header` argument can beused to specify “canned” policies and `put_bucket_policy` can be used to specify a more complex policy. The AWS Policy Generator can be useful for creating the appropriate JSON policy structure.

Value

For `get_policy`: A character string containing the JSON representation of the policy, if one has been set. For `delete_policy` and `put_policy`: `TRUE` if successful, `FALSE` otherwise.

References

API Documentation API Documentation AWS Policy Generator

`get_cors` *CORS*

Description

Get/Put/Delete the cross origin resource sharing configuration information for a bucket.

Usage

```
get_cors(bucket, ...)

put_cors(bucket, ...)

delete_cors(bucket, ...)
```

Arguments

`bucket`	Character string with the name of the bucket, or an object of class “s3_bucket”.
`...`	Additional arguments passed to `s3HTTP`.

Value

For `get_cors`: A list with cors configuration and rules. For `delete_cors`: `TRUE` if successful, `FALSE` otherwise.

References

API Documentation: PUT cors API Documentation: GET cords API Documentation: DELETE cors

`get_encryption` *Bucket encryption*

Description

Get/Put/Delete bucket-level encryption settings.

Usage

```
get_encryption(bucket, ...)

put_encryption(bucket, algorithm = c("AES256", "KMS"), kms_arn = NULL, ...)

delete_encryption(bucket, ...)
```

Arguments

`bucket`	Character string with the name of the bucket, or an object of class “s3_bucket”.
`...`	Additional arguments passed to `s3HTTP`.
`algorithm`	A character string specifying whether to use “AES256” or “KMS” encryption.
`kms_arn`	If `algorithm = "KMS"`, a KMS ARN.

Details

`get_encryption` returns the default encryption of a bucket; `put_encryption` sets the default encryption. `delete_encryption` deletes the encryption status.

Value

For `get_encryption`: if encryption has never been set, the value is `NULL`. Otherwise, the encryption type is returned as a charater string. For `put_encryption` or `delete_encryption`: a logical `TRUE`

References

API Documentation API Documentation API Documentation

Examples

```
## Not run:
 # example bucket
 put_bucket("mybucket")

 # set and check encryption
 put_encryption("mybucket", "AES256")
 get_encryption("mybucket")

 # delete encryption
 delete_encryption("mybucket")

## End(Not run)
```

`get_lifecycle` *Lifecycle*

Description

Get/Put/Delete the lifecycle configuration information for a bucket.

Usage

```
get_lifecycle(bucket, ...)

put_lifecycle(bucket, request_body, ...)

delete_lifecycle(bucket, ...)
```

Arguments

`bucket`	Character string with the name of the bucket, or an object of class "s3_bucket".
`...`	Additional arguments passed to `s3HTTP`.
`request_body`	A character string containing an XML request body, as defined in the specification in the API Documentation.

Value

For `get_lifecycle`: a list with lifecycle configuration, if it has been configured. For `delete_lifecycle`: `TRUE` if successful, `FALSE` otherwise.

References

API Documentation: PUT lifecycle API Documentation: GET lifecycle API Documentation: DELETE lifecycle

---

`get_location` *Bucket location*

Description

Get the AWS region location of bucket.

Usage

```
get_location(bucket, ...)
```

Arguments

`bucket`	Character string with the name of the bucket, or an object of class "s3_bucket".
`...`	Additional arguments passed to `s3HTTP`.

Value

A character string containing the region, if one has been set.

References

API Documentation

`get_notification` *Notifications*

Description

Get/put the notification configuration for a bucket.

Usage

```
get_notification(bucket, ...)

put_notification(bucket, request_body, ...)
```

Arguments

`bucket`	Character string with the name of the bucket, or an object of class "s3_bucket".
`...`	Additional arguments passed to `s3HTTP`.
`request_body`	A character string containing an XML request body, as defined in the specification in the API Documentation.

Value

A list containing the notification configuration, if one has been set.

References

API Documentation: GET API Documentation: PUT

`get_object` *Get object*

Description

Retrieve an object from an S3 bucket. To check if an object exists, see `head_object`

Usage

```
get_object(
  object,
  bucket,
  headers = list(),
  parse_response = FALSE,
  as = "raw",
  ...
)
```

```
save_object(
  object,
  bucket,
  file = basename(object),
  headers = list(),
  overwrite = TRUE,
  ...
)

select_object(
  object,
  bucket,
  request_body,
  headers = list(),
  parse_response = FALSE,
  ...
)

s3connection(object, bucket, headers = list(), ...)
```

Arguments

object	Character string with the object key, or an object of class "s3_object". In most cases, if object is specified as the latter, bucket can be omitted because the bucket name will be extracted from "Bucket" slot in object.
bucket	Character string with the name of the bucket, or an object of class "s3_bucket".
headers	List of request headers for the REST call.
parse_response	Passed through to s3HTTP, as this function requires a non-default setting. There is probably no reason to ever change this.
as	Passed through to httr::content.
...	Additional arguments passed to s3HTTP.
file	An R connection, or file name specifying the local file to save the object into.
overwrite	A logical indicating whether to overwrite file. Passed to write_disk. Default is TRUE.
request_body	For select_object, an XML request body as described in the SELECT API documentation.

Details

get_object retrieves an object into memory as a raw vector. This page describes get_object and several wrappers that provide additional useful functionality.

save_object saves an object to a local file without bringing it into memory.

s3connection provides a connection interface to an S3 object.

select_object uses the SELECT API to select part of a CSV or JSON object. This requires constructing and passing a fairly tedious request body, which users will have to construct themselves according to the documentation.

Some users may find the raw vector response format of get_object unfamiliar. The object will also carry attributes, including "content-type", which may be useful for deciding how to subsequently process the vector. Two common strategies are as follows. For text content types, running charToRaw may be the most useful first step to make the response human-readable. Alternatively, converting the raw vector into a connection using rawConnection may also be useful, as that can often then be passed to parsing functions just like a file connection would be.

Higher-level functions

Value

If file = NULL, a raw object. Otherwise, a character string containing the file name that the object is saved to.

References

API Documentation: GET Object API Documentation: GET Object torrent API Documentation: SELECT Object

See Also

get_bucket, object_exists, head_object, put_object, delete_object

Examples

```
## Not run:
  # get an object in memory
  ## create bucket
  b <- put_bucket("myexamplebucket")

  ## save a dataset to the bucket
  s3save(mtcars, bucket = b, object = "mtcars")
  obj <- get_bucket(b)
  ## get the object in memory
  x <- get_object(obj[[1]])
  load(rawConnection(x))
  "mtcars" %in% ls()

  # save an object locally
  y <- save_object(obj[[1]], file = object[[1]][["Key"]])
  y %in% dir()

  # return object using 'S3 URI' syntax, with progress bar
  get_object("s3://myexamplebucket/mtcars", show_progress = TRUE)

  # return parts of an object
  ## use 'Range' header to specify bytes
  get_object(object = obj[[1]], headers = list('Range' = 'bytes=1-120'))
  # example of streaming connection
  ## setup a bucket and object
  b <- put_bucket("myexamplebucket")
  s3write_using(mtcars, bucket = b, object = "mtcars.csv", FUN = utils::write.csv)

  ## setup the connection
  con <- s3connection("mtcars.csv", bucket = b)

  ## line-by-line read
  while(length(x <- readLines(con, n = 1L))) {
    print(x)
  }

  ## use data.table::fread without saving object to file
  library(data.table)
 s3write_using(as.data.table(mtcars), bucket = b, object = "mtcars2.csv", FUN = data.table::fwrite)
  fread(get_object("mtcars2.csv", bucket = b, as = "text"))

  ## cleanup
  close(con)
  delete_bucket("myexamplebucket")

## End(Not run)
```

get_replication *Bucket replication*

Description

Get/Delete the replication configuration for a bucket.

Usage

```
get_replication(bucket, ...)

put_replication(bucket, request_body, ...)

delete_replication(bucket, ...)
```

Arguments

bucket	Character string with the name of the bucket, or an object of class "s3_bucket".
...	Additional arguments passed to s3HTTP.
request_body	A character string containing an XML request body, as defined in the specification in the API Documentation.

Details

get_replication gets the current replication policy. delete_replication deletes the replication policy for a bucket.

Value

For `get_replication`: A list containing the replication configuration, if one has been set. For `delete_replication`: `TRUE` if successful, `FALSE` otherwise.

References

API Documentation: PUT replication API Documentation: GET replication API Documentation: DELETE replication

`get_requestpayment` *requestPayment*

Description

Get/Put the requestPayment subresource for a bucket.

Usage

```
get_requestpayment(bucket, ...)

put_requestpayment(bucket, ...)
```

Arguments

`bucket`	Character string with the name of the bucket, or an object of class "s3_bucket".
`...`	Additional arguments passed to `s3HTTP`.

Value

A list containing the requestPayment information, if set.

References

API Documentation

get_tagging *Bucket tagging*

Description

Get/delete the tag set for a bucket.

Usage

```
get_tagging(bucket, ...)

put_tagging(bucket, tags = list(), ...)

delete_tagging(bucket, ...)
```

Arguments

bucket	Character string with the name of the bucket, or an object of class "s3_bucket".
...	Additional arguments passed to s3HTTP.
tags	A list containing key-value pairs of tag names and values.

Value

A list containing the tag set, if one has been set. For delete_tagging: TRUE if successful, FALSE otherwise.

References

API Documentation: PUT tagging API Documentation: GET tagging API Documentation: DELETE tagging

Examples

```
## Not run: 
 put_tagging("mybucket", tags = list(foo = "1", bar = "2"))
 get_tagging("mybucket")
 delete_tagging("mybucket")

## End(Not run)
```

`get_torrent` *Get object torrent*

Description

Retrieves a Bencoded dictionary (BitTorrent) for an object from an S3 bucket.

Usage

```
get_torrent(object, bucket, ...)
```

Arguments

`object`	Character string with the object key, or an object of class "s3_object". In most cases, if `object` is specified as the latter, `bucket` can be omitted because the bucket name will be extracted from "Bucket" slot in `object`.
`bucket`	Character string with the name of the bucket, or an object of class "s3_bucket".
`...`	Additional arguments passed to `s3HTTP`.

Value

Something.

References

API Documentation

`get_uploads` *Multipart uploads*

Description

Get a list of multipart uploads for a bucket.

Usage

```
get_uploads(bucket, ...)
```

Arguments

`bucket`	Character string with the name of the bucket, or an object of class "s3_bucket".
`...`	Additional arguments passed to `s3HTTP`.

Value

A list containing the multipart upload information.

References

API Documentation

`get_versions` *Bucket versions*

Description

Get/Put versioning settings or retrieve versions of bucket objects.

Usage

```
get_versions(bucket, ...)

get_versioning(bucket, ...)

put_versioning(bucket, status = c("Enabled", "Suspended"), ...)
```

Arguments

`bucket`	Character string with the name of the bucket, or an object of class “s3_bucket”.
`...`	Additional arguments passed to `s3HTTP`.
`status`	Character string specifying whether versioning should be “Enabled” or “Suspended”.

Details

`get_versioning` returns the versioning status of a bucket; `put_versioning` sets the versioning status. `get_versions` returns information about bucket versions.

Value

For `get_versioning`: If versioning has never been enabled or suspend, the value is `NULL`. Otherwise, the status is returned (either “Enabled” or “Suspended”). For `put_versioning`: If versioning has never been enabled or suspend, the value is `NULL`. Otherwise, the status is returned (either “Enabled” or “Suspended”). For `get_versions`: A list.

References

API Documentation API Documentation API Documentation

Examples

```
## Not run:
 put_versioning("mybucket")
 get_versioning("mybucket")
 get_versions("mybucket")

## End(Not run)
```

`head_object` *Get object metadata*

Description

Check if an object from an S3 bucket exists. To retrieve the object, see `get_object`

Usage

```
head_object(object, bucket, ...)

object_exists(object, bucket, ...)

object_size(object, bucket, ...)
```

Arguments

`object`	Character string with the object key, or an object of class "s3_object". In most cases, if `object` is specified as the latter, `bucket` can be omitted because the bucket name will be extracted from "Bucket" slot in `object`.
`bucket`	Character string with the name of the bucket, or an object of class "s3_bucket".
`...`	Additional arguments passed to `s3HTTP`.

Details

`head_object` is a low-level API wrapper that checks whether an object exists by executing an HTTP HEAD request; this can be useful for checking object headers such as "content-length" or "content-type". `object_exists` is sugar that returns only the logical.

`object_size` returns the size of the object (from the "content-length" attribute returned by `head_object`).

Value

`head_object` returns a logical. `object_exists` returns `TRUE` if bucket exists and is accessible, else `FALSE`. `object_size` returns an integer, which is `NA` if the request fails.

References

API Documentation: HEAD Object

See Also

bucket_exists, get_object, put_object, delete_object

Examples

```
## Not run:
  # get an object in memory
  ## create bucket
  b <- put_bucket("myexamplebucket")

  ## save a dataset to the bucket
  s3save(mtcars, bucket = b, object = "mtcars")

  # check that object exists
  object_exists("mtcars", "myexamplebucket")
  object_exists("s3://myexamplebucket/mtcars")

  # get the object's size
  object_size("s3://myexamplebucket/mtcars")

  # get the object
  get_object("s3://myexamplebucket/mtcars")

## End(Not run)
```

---

put_bucket *Create bucket*

Description

Creates a new S3 bucket.

Usage

```
put_bucket(
  bucket,
  region = Sys.getenv("AWS_DEFAULT_REGION"),
  acl = c("private", "public-read", "public-read-write", "aws-exec-read",
    "authenticated-read", "bucket-owner-read", "bucket-owner-full-control"),
  location_constraint = region,
  headers = list(),
  ...
)
```

Arguments

bucket	Character string with the name of the bucket, or an object of class “s3_bucket”.

`region`	A character string containing the AWS region. If missing, defaults to value of environment variable `AWS_DEFAULT_REGION`.
`acl`	A character string indicating a "canned" access control list. By default all bucket contents and objects therein are given the ACL "private". This can later be viewed using `get_acl` and modified using `put_acl`.
`location_constraint`	A character string specifying a location constraint. If `NULL` (for example, for S3-compatible storage), no LocationConstraint body is passed.
`headers`	List of request headers for the REST call.
`...`	Additional arguments passed to `s3HTTP`.

Details

Bucket policies regulate who has what access to a bucket and its contents. The `header` argument can beused to specify "canned" policies and `put_bucket_policy` can be used to specify a more complex policy. The AWS Policy Generator can be useful for creating the appropriate JSON policy structure.

Value

`TRUE` if successful.

References

API Documentation AWS Policy Generator

See Also

`bucketlist`, `get_bucket`, `delete_bucket`, `put_object`, `put_encryption`, `put_versioning`

Examples

```
## Not run:
  put_bucket("examplebucket")

  # set a "canned" ACL to, e.g., make bucket publicly readable
  put_bucket("examplebucket", headers = list(`x-amz-acl` = "public-read"))


## End(Not run)
```

put_object *Put object*

Description

Puts an object into an S3 bucket

Usage

```
put_object(
  file,
  object,
  bucket,
  multipart = FALSE,
  acl = NULL,
  headers = list(),
  verbose = getOption("verbose", FALSE),
  show_progress = getOption("verbose", FALSE),
  ...
)

put_folder(folder, bucket, ...)
```

Arguments

`file`	A character string containing the filename (or full path) of the file you want to upload to S3. Alternatively, an raw vector containing the file can be passed directly, in which case `object` needs to be specified explicitly.
`object`	A character string containing the name the object should have in S3 (i.e., its "object key"). If missing, the filename is used.
`bucket`	Character string with the name of the bucket, or an object of class "s3_bucket".
`multipart`	A logical indicating whether to use multipart uploads. See `http://docs.aws.amazon.com/AmazonS3/latest/dev/mpuoverview.html`. If `file` is less than 100 MB, this is ignored.
`acl`	A character string indicating a "canned" access control list. By default all bucket contents and objects therein are given the ACL "private". This can later be viewed using `get_acl` and modified using `put_acl`.
`headers`	List of request headers for the REST call. If `multipart` = `TRUE`, this only applies to the initialization call.
`verbose`	A logical indicating whether to be verbose. Default is given by `options("verbose")`.
`show_progress`	A logical indicating whether to show a progress bar for uploads. Default is given by `options("verbose")`.
`...`	Additional arguments passed to `s3HTTP`.
`folder`	A character string containing a folder name. (A trailing slash is not required.)

Details

This provide a generic interface for sending files (or serialized, in-memory representations thereof) to S3. Some convenience wrappers are provided for common tasks: e.g., `s3save` and `s3saveRDS`.

Note that S3 is a flat file store. So there is no folder hierarchy as in a traditional hard drive. However, S3 allows users to create pseudo-folders by prepending object keys with `foldername/`. The `put_folder` function is provided as a high-level convenience function for creating folders. This is not actually necessary as objects with slashes in their key will be displayed in the S3 web console as if they were in folders, but it may be useful for creating an empty directory (which is possible in the web console).

Value

If successful, `TRUE`.

References

API Documentation

See Also

`put_bucket`, `get_object`, `delete_object`, `put_encryption`

Examples

```
## Not run:
  library("datasets")

  # write file to S3
  tmp <- tempfile()
  on.exit(unlink(tmp))
  utils::write.csv(mtcars, file = tmp)
  # put object with an upload progress bar
 put_object(tmp, object = "mtcars.csv", bucket = "myexamplebucket", show_progress = TRUE)

  # create a "folder" in a bucket
  put_folder("example", bucket = "myexamplebucket")
  ## write object to the "folder"
  put_object(tmp, object = "example/mtcars.csv", bucket = "myexamplebucket")

  # write serialized, in-memory object to S3
  x <- rawConnection(raw(0), "w")
  utils::write.csv(mtcars, x)
 put_object(rawConnectionValue(x), object = "mtcars.csv", bucket = "myexamplebucketname")

  # use `headers` for server-side encryption
  ## require appropriate bucket policy
  ## encryption can also be set at the bucket-level using \code{\link{put_encryption}}
  put_object(file = tmp, object = "mtcars.csv", bucket = "myexamplebucket",
             headers = c('x-amz-server-side-encryption' = 'AES256'))

  # alternative "S3 URI" syntax:
```

```
  put_object(rawConnectionValue(x), object = "s3://myexamplebucketname/mtcars.csv")
  close(x)

  # read the object back from S3
  read.csv(text = rawToChar(get_object(object = "s3://myexamplebucketname/mtcars.csv")))

  # multi-part uploads for objects over 5MB
  \donttest{
  x <- rnorm(3e6)
  saveRDS(x, tmp)
  put_object(tmp, object = "rnorm.rds", bucket = "myexamplebucket",
             show_progress = TRUE, multipart = TRUE)
  identical(x, s3readRDS("s3://myexamplebucket/rnorm.rds"))
  }

## End(Not run)
```

s3HTTP *S3 HTTP Requests*

Description

This is the workhorse function for executing API requests for S3.

Usage

```
s3HTTP(
  verb = "GET",
  bucket = "",
  path = "",
  query = NULL,
  headers = list(),
  request_body = "",
  write_disk = NULL,
  write_fn = NULL,
  accelerate = FALSE,
  dualstack = FALSE,
  parse_response = TRUE,
  check_region = FALSE,
  url_style = c("path", "virtual"),
  base_url = Sys.getenv("AWS_S3_ENDPOINT", "s3.amazonaws.com"),
  verbose = getOption("verbose", FALSE),
  show_progress = getOption("verbose", FALSE),
  region = NULL,
  key = NULL,
  secret = NULL,
  session_token = NULL,
  use_https = TRUE,
  ...
)
```

Arguments

verb	A character string containing an HTTP verb, defaulting to "GET".
bucket	A character string with the name of the bucket, or an object of class "s3_bucket". If the latter and a region can be inferred from the bucket object attributes, then that region is used instead of `region`.
path	A character string with the name of the object to put in the bucket (sometimes called the object or 'key name' in the AWS documentation.)
query	Any query arguments, passed as a named list of key-value pairs.
headers	A list of request headers for the REST call.
request_body	A character string containing request body data.
write_disk	If `verb = "GET"`, this is, optionally, an argument like `write_disk` to write the result directly to disk.
write_fn	If set to a function and `verb = "GET"` is used then the output is passed in chunks as a raw vector in the first argument to this function, allowing streaming output. Note that `write_disk` and `write_fn` are mutually exclusive.
accelerate	A logical indicating whether to use AWS transfer acceleration, which can produce significant speed improvements for cross-country transfers. Acceleration only works with buckets that do not have dots in bucket name.
dualstack	A logical indicating whether to use "dual stack" requests, which can resolve to either IPv4 or IPv6. See http://docs.aws.amazon.com/AmazonS3/latest/dev/dual-stack-endpoints.html.
parse_response	A logical indicating whether to return the response as is, or parse and return as a list. Default is `TRUE`.
check_region	A logical indicating whether to check the value of `region` against the apparent bucket region. This is useful for avoiding (often confusing) out-of-region errors. Default is `FALSE`.
url_style	A character string specifying either "path" (the default), or "virtual"-style S3 URLs.
base_url	A character string specifying the base hostname for the request (it is a misnomer, the actual URL is constructed from this name, region and `use_https` flag. There is no need to set this, as it is provided only to generalize the package to (potentially) support S3-compatible storage on non-AWS servers. The easiest way to use S3-compatible storage is to set the `AWS_S3_ENDPOINT` environment variable. When using non-AWS servers, you may also want to set `region=""`.
verbose	A logical indicating whether to be verbose. Default is given by `options("verbose")`.
show_progress	A logical indicating whether to show a progress bar for downloads and uploads. Default is given by `options("verbose")`.
region	A character string containing the AWS region. Ignored if region can be inferred from `bucket`. If missing, an attempt is made to locate it from credentials. Defaults to "us-east-1" if all else fails. Should be set to `""` when using non-AWS endpoints that don't include regions (and `base_url` must be set).

`key`	A character string containing an AWS Access Key ID. If missing, defaults to value stored in environment variable `AWS_ACCESS_KEY_ID`.
`secret`	A character string containing an AWS Secret Access Key. If missing, defaults to value stored in environment variable `AWS_SECRET_ACCESS_KEY`.
`session_token`	Optionally, a character string containing an AWS temporary Session Token. If missing, defaults to value stored in environment variable `AWS_SESSION_TOKEN`.
`use_https`	Optionally, a logical indicating whether to use HTTPS requests. Default is `TRUE`.
`...`	Additional arguments passed to an HTTP request function. such as `GET`.

Details

This is mostly an internal function for executing API requests. In almost all cases, users do not need to access this directly.

Value

the S3 response, or the relevant error.

`s3save` *save/load*

Description

Save/load R object(s) to/from S3

Usage

```
s3save(..., object, bucket, envir = parent.frame(), opts = NULL)

s3save_image(object, bucket, opts = NULL)

s3load(object, bucket, envir = parent.frame(), ...)
```

Arguments

`...`	For `s3save`, one or more R objects to be saved via `save` and uploaded to S3. For `s3load`, see `opts`.
`object`	For `s3save`, a character string of the name of the object you want to save to. For `s3load`, a character string of the name of the object you want to load from S3.
`bucket`	Character string with the name of the bucket, or an object of class “s3_bucket”.
`envir`	For `s3save`, an R environment to save objects from; for `s3load`, the environment to load objects into. Default is the `parent.frame()` from which the function is called.
`opts`	Additional arguments passed to `s3HTTP`.

Value

For `s3save`, a logical, invisibly. For `s3load`, `NULL` invisibly.

References

API Documentation

See Also

`s3saveRDS`,`s3readRDS`

Examples

```
## Not run:
# create bucket
b <- put_bucket("myexamplebucket")

# save a dataset to the bucket
s3save(mtcars, iris, object = "somedata.Rdata", bucket = b)
get_bucket(b)

# load the data from bucket
e <- new.env()
s3load(object = "somedata.Rdata", bucket = b, envir = e)
ls(e)

# cleanup
rm(e)
delete_object(object = "somedata.Rdata", bucket = "myexamplebucket")
delete_bucket("myexamplebucket")

## End(Not run)
```

`s3saveRDS` *saveRDS/readRDS*

Description

Serialization interface to read/write R objects to S3

Usage

```
s3saveRDS(
  x,
  object = paste0(as.character(substitute(x)), ".rds"),
  bucket,
  compress = TRUE,
  ...
)
```

```
s3readRDS(object, bucket, ...)
```

Arguments

x	For s3saveRDS, a single R object to be saved via saveRDS and uploaded to S3. x is analogous to the object argument in saveRDS.
object	Character string with the object key, or an object of class "s3_object". In most cases, if object is specified as the latter, bucket can be omitted because the bucket name will be extracted from "Bucket" slot in object.
bucket	Character string with the name of the bucket, or an object of class "s3_bucket".
compress	A logical. See saveRDS.
...	Additional arguments passed to s3HTTP.

Details

Note that early versions of s3saveRDS from aws.s3 <= 0.2.4 unintentionally serialized objects to big endian format (due to defaults in serialize. This can create problems when attempting to read these files using readRDS. The function attempts to catch the issue and read accordingly, but may fail. The solution used internally is `unserialize(memDecompress(get_object(),"gzip"))`

Value

For s3saveRDS, a logical. For s3readRDS, an R object.

Author(s)

Steven Akins <skawesome@gmail.com>

See Also

s3save,s3load

Examples

```
## Not run:
# create bucket
b <- put_bucket("myexamplebucket")

# save a single object to s3
s3saveRDS(x = mtcars, bucket = "myexamplebucket", object = "mtcars.rds")

# restore it under a different name
mtcars2 <- s3readRDS(object = "mtcars.rds", bucket = "myexamplebucket")
identical(mtcars, mtcars2)

# cleanup
delete_object(object = "mtcars.rds", bucket = "myexamplebucket")
delete_bucket("myexamplebucket")

## End(Not run)
```

`s3source` *Source from S3*

Description

Source R code (a la `source`) from S3

Usage

```
s3source(object, bucket, ..., opts = NULL)
```

Arguments

`object`	Character string with the object key, or an object of class "s3_object". In most cases, if `object` is specified as the latter, `bucket` can be omitted because the bucket name will be extracted from "Bucket" slot in `object`.
`bucket`	Character string with the name of the bucket, or an object of class "s3_bucket".
`...`	Additional arguments passed to `s3HTTP`.
`opts`	Additional arguments passed to `get_object` for retrieving the R syntax file.

Value

See `source`

See Also

`s3saveRDS`,`s3save`, `get_object`

Examples

```
## Not run:
# create bucket
b <- put_bucket("myexamplebucket")

# save some code to the bucket
cat("x <- 'hello world!'\nx", file = "example.R")
put_object("example.R", object = "example.R", bucket = b)
get_bucket(b)

# source the code from the bucket
s3source(object = "example.R", bucket = b, echo = TRUE)

# cleanup
unlink("example.R")
delete_object(object = "example.R", bucket = b)
delete_bucket("myexamplebucket")

## End(Not run)
```

s3sync *S3 file sync*

Description

Sync files/directories to/from S3

Usage

```
s3sync(
  path = ".",
  bucket,
  prefix = "",
  direction = c("upload", "download"),
  verbose = TRUE,
  create = FALSE,
  ...
)
```

Arguments

`path`	string, path to the directory to synchronize, it will be expanded as needed (NOTE: older versions had a `files` argument which expected a full list of files which was ambiguous).
`bucket`	Character string with the name of the bucket, or an object of class "s3_bucket".
`prefix`	string, if set to non-empty string, leading part of the objects in the bucket much have that prefix, other objects are not considered. In practice, this alows the immitation of sub-directories in the bucket and in that case it is typically required that the training slash is included in the prefix.
`direction`	A character vector specifying whether to "upload" and/or "download" files. By default, `s3sync` is two-way, uploading any files missing from the bucket and downloading any objects missing from the local directory.
`verbose`	A logical indicating whether to be verbose (the default is `TRUE`).
`create`	logical, if `TRUE` the bucket is created if it doesn't exist, otherwise synchronizing a non-existing bucket is an error.
`...`	Additional arguments passed to `s3HTTP`.

Details

`s3sync` synchronizes specified files to an S3 bucket. If the bucket does not exist, it is created (unless `create=FALSE`). Similarly, if local directories do not exist (corresponding to leading portions of object keys), they are created, recursively. Object keys are generated based on `files` and local files are named (and organized into directories) based on object keys. A slash is interpreted as a directory level. Local objects are copied to S3 and S3 objects are copied locally. This copying is performed conditionally. Objects existing locally but not in S3 are uploaded using `put_object`. Objects existing in S3 but not locally, are saved using `save_object`. If objects exist in both places, the MD5 checksum for each is compared; when identical, no copying is performed. If the checksums differ, local files are replaced with the bucket version if the local file is older and the S3 object is replaced if the local file is newer. If checksums differ but modified times match (which seems unlikely), a warning is issued. Note that multi-part files don't have a full MD5 sum recorded in S3 so they cannot be compared and thus are always assumed to be different.

Value

A logical.

References

aws s3 sync command line

See Also

`get_bucket`, `put_object`, , `save_object`

Examples

```
## Not run:
  put_bucket("examplebucket")

  # sync all files in current directory to bucket (upload-only)
  s3sync(bucket = "examplebucket", direction = "upload")

  # two-way sync
  s3sync(bucket = "examplebucket")

  # full sync between a subset of the bucket and a test directory in user's home
  # corresponding roughly to:
  #   aws s3 sync ~/test s3://examplebucket/test/
  #   aws s3 sync s3://examplebucket/test/ ~/test
  s3sync("~/test", "examplebucket", prefix="test/", region="us-east-2")

## End(Not run)
```

`s3write_using` *Custom read and write*

Description

Read/write objects from/to S3 using a custom function

Usage

```
s3write_using(x, FUN, ..., object, bucket, opts = NULL)

s3read_using(FUN, ..., object, bucket, opts = NULL, filename = NULL)
```

Arguments

x	For s3write_using, a single R object to be saved via the first argument to FUN and uploaded to S3.
FUN	For s3write_using, a function to which x and a file path will be passed (in that order).
...	Additional arguments to FUN
object	Character string with the object key, or an object of class "s3_object". In most cases, if object is specified as the latter, bucket can be omitted because the bucket name will be extracted from "Bucket" slot in object.
bucket	Character string with the name of the bucket, or an object of class "s3_bucket".
opts	Optional additional arguments passed to put_object or save_object, respectively.
filename	Optional string, name of the temporary file that will be created. If not specified, tempfile() with the extension of the object is used.

Value

For s3write_using, a logical, invisibly. For s3read_using, the output of FUN applied to the file from object.

See Also

s3saveRDS, s3readRDS, put_object,get_object

Examples

```
## Not run:
library("datasets")
# create bucket
b <- put_bucket("myexamplebucket")

# save a dataset to the bucket as a csv
if (require("utils")) {
  s3write_using(mtcars, FUN = write.csv, object = "mtcars.csv", bucket = b)
}

# load dataset from the bucket as a csv
if (require("utils")) {
  s3read_using(FUN = read.csv, object = "mtcars.csv", bucket = b)
}

# cleanup
delete_object(object = "mtcars.csv", bucket = b)
delete_bucket(bucket = b)

## End(Not run)
```

Index