Package 'paws.common'

February 17, 2026

Type Package

Title Paws Low-Level Amazon Web Services API

Version 0.8.9

Description Functions for making low-level API requests to Amazon Web Services <https://aws.amazon.com>. The functions handle building, signing, and sending requests, and receiving responses. They are designed to help build higher-level interfaces to individual services, such as Simple Storage Service (S3).

License Apache License (>= 2.0)

Depends R (>= 4.1.0)

URL https://github.com/paws-r/paws, https://paws-r.r-universe.dev/paws.common, https://www.paws-r-sdk.com

BugReports https://github.com/paws-r/paws/issues

Encoding UTF-8

LinkingTo Rcpp

Imports base64enc, curl, digest, httr2 (>= 1.0.4), jsonlite, methods, utils, stats, Rcpp, xml2

Suggests covr, crayon, mockery, withr, rstudioapi, testthat (>= 3.0.0)

SystemRequirements pandoc (>= 1.12.3) - http://pandoc.org

RoxygenNote 7.3.3

Collate 'RcppExports.R' 'util.R' 'cache.R' 'struct.R' 'handlers.R' 'iniutil.R' 'config.R' 'logging.R' 'dateutil.R' 'credential_sso.R' 'credential_sts.R' 'url.R' 'net.R' 'credential_providers.R' 'credentials.R' 'client.R' 'convert.R' 'service.R' 'custom_dynamodb.R' 'custom_rds.R' 'head_bucket.R' 'http_status.R' 'error.R' 'custom_s3.R' 'handlers_core.R' 'handlers_ec2query.R' 'handlers_jsonrpc.R' 'handlers_query.R' 'handlers_rest.R' 'handlers_restjson.R' 'tags.R' 'xmlutil.R' 'handlers_restxml.R' 'handlers_stream.R' 'idempotency.R'

'jsonutil.R' 'mock_bindings.R' 'onLoad.R' 'paginate.R'
'populateutil.R' 'queryutil.R' 'request.R' 'retry.R'
'service_parameter_helper.R' 'signer_bearer.R' 'signer_v4.R'
'signer_s3.R' 'signer_s3v4.R' 'signer_v1.R' 'signer_v2.R'
'time.R'

Config/testthat/edition 3

NeedsCompilation yes

Author David Kretch [aut],
Adam Banker [aut],
Dyfan Jones [cre],
Amazon.com, Inc. [cph]

Maintainer Dyfan Jones <dyfan.r.jones@gmail.com>

Repository CRAN

Date/Publication 2026-02-17 06:10:20 UTC

Contents

as.list.struct *Create a list from an struct object*

Description

Create a list from an struct object

Usage

```
## S3 method for class 'struct'
as.list(x, ...)
```

Arguments

x	An struct object.
...	Other arguments, which will be ignored.

get_config *Get the service configuration from the service object.*

Description

Look up the service configuration from the service object, e.g. when calling `svc$operation()`, `get_config()` will look up `svc`, then get any configuration stored in it, as if the operation function were a method and the service object were a class instance.

Usage

```
get_config()
```

Details

`get_config` must be called directly by the operation function and assigned immediately, not provided as an argument to another function.

We look up the service object then fetch its data so we can both support documentation tooltips in RStudio and also have class-object-like behavior. Alternatives that do not support documentation tooltips in RStudio include reference classes (RC), R6 classes, and any modification of the functions at run-time, e.g. inserting the configuration into the function definition for each operation in a particular service object.

is_empty *Check whether an object is empty*

Description

Check whether an object is empty, e.g. has no sub-elements, is NA, or is the empty string.

Usage

```
is_empty(x)
```

Arguments

x	An object.

Examples

```
is_empty(NA) # TRUE
is_empty("") # TRUE
is_empty(list()) # TRUE
is_empty(list(list())) # TRUE

is_empty(1) # FALSE
is_empty(list(1)) # FALSE
is_empty(list(list(1))) # FALSE
```

is_empty_xml *Check whether an object is empty for xml builds*

Description

Check whether an object is empty, e.g. has no sub-elements, is NA

Usage

```
is_empty_xml(x)
```

Arguments

x	An object.

Examples

```
is_empty_xml(NA) # TRUE
is_empty_xml(list()) # TRUE
is_empty_xml(list(list())) # TRUE

is_empty_xml(1) # FALSE
is_empty_xml("") # FALSE
is_empty_xml(list(1)) # FALSE
is_empty_xml(list(list(1))) # FALSE
```

`list_paginators` *List methods that can be paginated from a paws client.*

Description

List methods that can be paginated from a paws client.

Usage

```
list_paginators(svc)
```

Arguments

`svc`	paws client (for example `paws::s3()`).

Value

character vector of functions that can be paginated.

Examples

```
## Not run:
# Note: where svc is a paws client.
list_paginators(svc)

## End(Not run)
```

locate_credentials *Locate AWS credentials*

Description

Locate AWS credentials

Usage

```
locate_credentials(profile = "", anonymous = FALSE)
```

Arguments

profile	The name of a profile to use. If not given, then the default profile is used.
anonymous	Set anonymous credentials.

Value

list containing AWS credentials

- access_key_id - (character) AWS access key ID
- secret_access_key - (character) AWS secret access key
- session_token - (character) AWS temporary session token
- access_token - (character) A token that gives a user permission to access certain resources
- expiration - (numeric) Indicates the Unix time when an access token will expire.
- region - (character) The AWS Region used in instantiating the client.

new_handlers *Return request handlers for a service*

Description

Return request handlers for a given protocol and request signer.

Usage

```
new_handlers(protocol, signer)
```

Arguments

protocol	Protocol: ec2query, jsonrpc, query, rest, restjson, or restxml.
signer	Signer: v2 or v4.

See Also

Other API request functions: new_operation(), new_request(), new_service(), send_request()

Examples

```
# Get the handlers needed for an API using REST-JSON and AWS signature V4.
handlers <- new_handlers("restjson", "v4")
```

`new_operation` *Return an API operation object*

Description

Return an API operation object, with information on what to request for a given API operation. For example, the S3 service's "list buckets" operation is named `ListBuckets`, it requires a `GET` request, and so on.

Usage

```
new_operation(
  name,
  http_method,
  http_path,
  host_prefix,
  paginator,
  stream_api = FALSE,
  before_presign_fn = NULL
)
```

Arguments

`name`	The API operation name.
`http_method`	The HTTP method, e.g. `"GET"` or `"POST"`.
`http_path`	The HTTP path.
`host_prefix`	The HTTP prefix
`paginator`	List input_token and output_token.
`stream_api`	Set if operation is stream api or not
`before_presign_fn`	Currently unused.

See Also

Other API request functions: new_handlers(), new_request(), new_service(), send_request()

Examples

```
# Save info about the S3 ListBuckets API operation.
op <- new_operation(
  name = "ListBuckets",
  http_method = "GET",
  http_path = "/",
  paginator = list()
)
```

`new_request` *Return an API request object*

Description

Return an API request object with everything needed to make a request.

Usage

```
new_request(client, operation, params, data, dest = NULL)
```

Arguments

`client`	A service client, e.g. from `new_service`.
`operation`	An operation, e.g. from `new_operation`.
`params`	A populated input object.
`data`	An empty output object.
`dest`	Control where the response body is written

See Also

Other API request functions: `new_handlers()`, `new_operation()`, `new_service()`, `send_request()`

Examples

```
## Not run:
# Make a request object for the S3 ListBuckets operation.
metadata <- list(
  endpoints = list("*" = list(endpoint = "s3.{region}.amazonaws.com", global = FALSE)),
  service_name = "s3"
)
client <- new_service(metadata, new_handlers("restxml", "s3"))
op <- new_operation("ListBuckets", "GET", "/", list())
params <- list()
data <- tag_add(list(Buckets = list()), list(type = "structure"))
req <- new_request(client, op, params, data)

## End(Not run)
```

new_service *Return an AWS API service object*

Description

Return an API service object with information and handlers needed to make API requests.

Usage

```
new_service(metadata, handlers, cfgs = NULL, operation = Operation())
```

Arguments

metadata	A named list of API metadata. It should look like:

```
list(
  service_name = "string",
 endpoints = list("region" = list(endpoint = "endpoint", global = FALSE)),
  service_id = "string",
  api_version = "string",
  signing_name = "string"|NULL,
  json_version = "string",
  target_prefix = "string"
)
```

handlers	A set of handlers, e.g. from new_handlers.
cfgs	A config defined by the service. Defaults to null.
operation	A operation defined by the service.

Region and credentials

new_service requires that you've set your AWS region in one of:

1. AWS_REGION R environment variable
2. AWS_REGION OS environment variable (Linux and macOS)
3. ~/.aws/config AWS configuration file

new_service also requires that you've set your AWS credentials in one of:

1. AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY R environment variables
2. AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY OS environment variables (Linux and macOS)
3. ~/.aws/credentials AWS credentials file
4. IAM role

See Also

new_operation()

Other API request functions: new_handlers(), new_operation(), new_request(), send_request()

Examples

```
## Not run:
# Metadata for the S3 API.
metadata <- list(
  service_name = "s3",
  endpoints = list("us-east-1" = list(endpoint = "s3.amazonaws.com", global = FALSE)),
  service_id = "S3",
  api_version = "2006-03-01",
  signing_name = NULL,
  json_version = "",
  target_prefix = ""
)

# Handlers for S3.
handlers <- new_handlers("restxml", "v4")

# Build a service object for S3, containing the information necessary to
# build, send, and receive requests.
service <- new_service(metadata, handlers)

## End(Not run)
```

---

paginate *Paginate over an operation.*

Description

Some AWS operations return results that are incomplete and require subsequent requests in order to attain the entire result set. The process of sending subsequent requests to continue where a previous request left off is called pagination. For example, the list_objects operation of Amazon S3 returns up to 1000 objects at a time, and you must send subsequent requests with the appropriate Marker in order to retrieve the next page of results.

Usage

```
paginate(
  Operation,
  PageSize = NULL,
  MaxItems = NULL,
  StartingToken = NULL,
  StopOnSameToken = FALSE
)
```

```
paginate_lapply(
  Operation,
  FUN,
  ...,
  PageSize = NULL,
  MaxItems = NULL,
  StartingToken = NULL,
  StopOnSameToken = FALSE
)

paginate_sapply(
  Operation,
  FUN,
  ...,
  simplify = TRUE,
  PageSize = NULL,
  MaxItems = NULL,
  StartingToken = NULL,
  StopOnSameToken = FALSE
)
```

Arguments

Operation	The operation for example an s3 operation: `svc$list_buckets()`
PageSize	The size of each page.
MaxItems	Limits the maximum number of total returned items returned while paginating.
StartingToken	Can be used to modify the starting marker or token of a paginator. This argument if useful for resuming pagination from a previous token or starting pagination at a known position.
StopOnSameToken	Exits paginator if previous token matches current token. For some APIs, such as CloudWatchLogs events, the next page token will always be present. When set to TRUE, the paginator will stop when the token doesn't change.
FUN	the function to be applied to each response element of `operation`.
...	optional arguments to FUN.
simplify	See base::sapply().

Value

list of responses from the operation.

Examples

```
## Not run:
# The following example retrieves object list. The request specifies max
# keys to limit response to include only 2 object keys.
```

```
paginate(
  svc$list_objects_v2(
    Bucket = "DOC-EXAMPLE-BUCKET"
  ),
  MaxItems = 50
)

## End(Not run)
```

`paws_config_log` *paws logging system*

Description

Ability to configure paws logging system, through the use of paws helper function `` `paws_config_log` `` or `R:base` `options` function. Users are able to change logging levels without calling `paws.common` by the use of `options` e.g. `options("paws.log_level" = 2L)`.

- `paws.log_level` (integer): The minimum log level that should be tracked
- `paws.log_file` (character): path for logs to populate, default output logs to console.
- `paws.log_timestamp_fmt` (character): see `format.POSIXct()`

Usage

```
paws_config_log(
  level = 2L,
  file = "",
  timestamp_fmt = "%Y-%m-%d %H:%M:%OS3"
)
```

Arguments

`level`	(integer) to determine the level logging threshold. • `5L`: TRACE • `4L`: DEBUG • `3L`: INFO • `2L`: WARNING • `1L`: ERROR
`file`	(character) path for logs to populate, default output logs to console.
`timestamp_fmt`	(character) for timestamp format, see `format.POSIXct()`.

Examples

```
# log to a file
temp_file <- tempfile()
paws_config_log(file = temp_file)
unlink(temp_file)

# change log threshold to INFO
paws_config_log(level = 3L)

# reset to default config
paws_config_log()
```

---

paws_reset_cache *Clear down paws cache environments*

Description

Clears down the cache environments.

- `ini_cache`: an environment that stores the results from `read_ini`, mainly used for storing AWS config files.
- `os_env_cache`: an environment that stores environmental variables from Unix Operating Systems.
- `bearer_token_cache`: an environment that stores bearer tokens

Usage

```
paws_reset_cache()
```

---

paws_stream *Iterate over AWS Event Stream connection*

Description

Iterate over AWS Event Stream connection

Usage

```
paws_stream_handler(FUN, .connection = FALSE)

paws_stream_parser(con)
```

Arguments

FUN	function to iterate over stream connection.
.connection	return paws_connection object a subclass of httr2::req_perform_connection (default FALSE)
con	A streaming response created by paws_stream_handler.

Value

list of responses from the operation or a paws_connection object

Examples

```
## Not run:
# Developed from:
# https://docs.aws.amazon.com/code-library/latest/ug/python_3_bedrock-runtime_code_examples.html
library(paws)

# Create a Bedrock Runtime client in the AWS Region you want to use.
client <- bedrockruntime(region = "us-east-1")

# Set the model ID, e.g., Titan Text Premier.
model_id <- "amazon.titan-text-premier-v1:0"

# Start a conversation with the user message.
user_message <- "Describe the purpose of a 'hello world' program in one line."
conversation <- list(
  list(
    role = "user",
    content = list(list(text = user_message))
  )
)

resp <- client$converse_stream(
  modelId = model_id,
  messages = conversation,
  inferenceConfig = list(maxTokens = 512, temperature = 0.5, topP = 0.9)
)
resp$stream(\(chunk) chunk$contentBlockDelta$delta$text)
# Note: stream will close connection after all chunks are read

# Get connection object
resp <- client$converse_stream(
  modelId = model_id,
  messages = conversation,
  inferenceConfig = list(maxTokens = 512, temperature = 0.5, topP = 0.9)
)
con <- resp$stream(.connection = T)

# Handle connection object using paws_stream_parser
while (!is.null(chunk <- paws_stream_parser(con))) {
  print(chunk$contentBlockDelta$delta$text)
```

```
}

# Note: paws_stream_parser will close connection after all chunks are read

resp <- client$converse_stream(
  modelId = model_id,
  messages = conversation,
  inferenceConfig = list(maxTokens = 512, temperature = 0.5, topP = 0.9)
)

# Or handle connection using httr2::resp_stream_aws
while (!is.null(chunk <- resp_stream_aws(con))) {
  str(chunk)
}
close(con)

# Note: connection needs to be closed manually after all chunks have been read.

## End(Not run)
```

`populate` *Populate a list with data from another list*

Description

`populate` copies data from a list (e.g. input by a user) to another list with a similar shape. The second list, called the `interface`, will generally also contain extra metadata for making API requests, such as names or types.

Usage

```
populate(input, interface, parent = NULL)
```

Arguments

`input`	A list with data to copy.
`interface`	A list of a similar shape to copy data into.
`parent`	Internal parameter used to track parent interface for recursive structures.

Examples

```
# Make an interface with metadata, e.g. type.
interface <- tag_add(list(foo = c(), bar = c()), list(type = "structure"))

# Combine data and the metadata from the interface.
populate(list(foo = 1, bar = 2), interface)
```

send_request *Send a request and handle the response*

Description

Send a request and handle the response. Build the HTTP request, send it to AWS, interpret the response, and throw an error if the response is not ok.

Usage

```
send_request(request)
```

Arguments

request	A request, e.g. from new_request.

See Also

Other API request functions: new_handlers(), new_operation(), new_request(), new_service()

Examples

```
## Not run:
# Send a request and handle the response.
resp <- send_request(req)

## End(Not run)
```

set_config *Add configuration settings to a service object.*

Description

set_config adds a given set of configuration settings in cfgs to a service object, i.e. the service object for S3. Configuration settings can include credentials, region, endpoint, etc. These configuration settings will be used whenever an operation is called from that service object.

Usage

```
set_config(svc, cfgs = list())
```

Arguments

svc	A service object containing service operations.
cfgs	A list of optional configuration settings.

Details

`set_config` explicitly makes the `credentials` property mutable, such that when the SDK retrieves credentials later on, it will save them in the service object. This means that credentials don't need to be fetched on each operation, only if and when the saved credentials expire.

The optional configuration settings can include the following:

```
list(
  credentials = list(
    creds = list(
      access_key_id = "string",
      secret_access_key = "string",
      session_token = "string"
    ),
    profile = "string"
  ),
  endpoint = "string",
  region = "string"
)
```

Examples

```
# Create a config object with custom credentials and endpoint.
config <- set_config(
  svc = list(),
  cfgs = list(
    credentials = list(
      creds = list(
        access_key_id = "abc",
        secret_access_key = "123"
      )
    ),
    endpoint = "https://foo.com"
  )
)
```

`set_service_parameter` *Set service parameters*

Description

Help functions for setting the parameters for services

Usage

```
config(
  credentials = list(creds = list(access_key_id = "", secret_access_key = "",
   session_token = "", access_token = "", expiration = Inf), profile = "", anonymous =
```

```
    FALSE),
  endpoint = "",
  region = "",
  close_connection = FALSE,
  max_retries = 3,
  connect_timeout = 60,
  s3_force_path_style = FALSE,
  s3_virtual_address = FALSE,
  sts_regional_endpoint = "",
  signature_version = ""
)

credentials(
  creds = list(access_key_id = "", secret_access_key = "", session_token = "",
    access_token = "", expiration = Inf),
  profile = "",
  anonymous = FALSE
)

creds(
  access_key_id = "",
  secret_access_key = "",
  session_token = "",
  access_token = "",
  expiration = Inf
)
```

Arguments

`credentials` `credentials()` or `list` in same format.

- **creds**: `creds()` or `list` in same format.
  - **access_key_id**: AWS access key ID
  - **secret_access_key**: AWS secret access key
  - **session_token**: AWS temporary session token
  - **access_token**: The token issued by the `CreateToken` API call. For more information, see CreateToken in the `IAM Identity Center OIDC API Reference Guide`.
  - **expiration**: The date and time when the temporary credentials expire. `expiration` must be a `POSIXct` date-time or able to be compared with them.
- **profile**: The name of a profile to use. If not given, then the default profile is used.
- **anonymous**: Set anonymous credentials.

`endpoint` The complete URL to use for the constructed client.

`region` The AWS Region used in instantiating the client.

`close_connection` Immediately close all HTTP connections.

`max_retries` Max number of retries call AWS API (default set to 3).

`connect_timeout`
: The time in seconds till a timeout exception is thrown when attempting to make a connection. The default is 60 seconds.

`s3_force_path_style`
: Set this to `true` to force the request to use path-style addressing, i.e. `http://s3.amazonaws.com/BUCKET`

`s3_virtual_address`
: Set this to `true` to force the request to use virtual-hosted-style

`sts_regional_endpoint`
: Set sts regional endpoint resolver to regional or legacy https://docs.aws.amazon.com/sdkref/latest/guide/feature-sts-regionalized-endpoints.html

`signature_version`
: The signature version used when signing requests. Note that the default version is Signature Version 4.

`creds` `creds()` or `list` in same format.

- **access_key_id**: AWS access key ID
- **secret_access_key**: AWS secret access key
- **session_token**: AWS temporary session token
- **access_token**: The token issued by the `CreateToken` API call. For more information, see CreateToken in the `IAM Identity Center OIDC API Reference Guide`.
- **expiration**: The date and time when the temporary credentials expire. `expiration` must be a `POSIXct` date-time or able to be compared with them.

`profile` The name of a profile to use. If not given, then the default profile is used.

`anonymous` Set anonymous credentials.

`access_key_id` AWS access key ID

`secret_access_key`
: AWS secret access key

`session_token` AWS temporary session token

`access_token` The token issued by the `CreateToken` API call. For more information, see CreateToken in the `IAM Identity Center OIDC API Reference Guide`.

`expiration` The date and time when the temporary credentials expire. `expiration` must be a `POSIXct` date-time or able to be compared with them.

Value

list set of parameter variables for paws services.

Examples

```
# set service parameter access_key_id and secret_access_key

config(credentials(creds("dummy", "secret")))

# set service parameter access_key_id and secret_access_key using using lists
```

```
config(
    credentials = list(
        creds = list(
            access_key_id = "dummy",
            secret_access_key = "secret"
        )
    )
)
```

---

tags *Get, set, and delete object tags*

Description

Tags are metadata stored in an object's attributes, used to store types and names needed to make AWS API requests.

`tag_get` returns the value of the given `tag`, or `""` if the tag doesn't exist.

`tag_has` returns whether the object has the given `tag`.

`tag_add` returns the object after adding the given list of tags and values.

`tag_del` returns the object after recursively deleting tags in `tags`, or all tags if `NULL`.

`type` returns broadly what type an object is, based on its `type` tag.

Usage

```
tag_get(object, tag)

tag_get_all(object)

tag_has(object, tag)

tag_add(object, tags)

tag_del(object, tags = NULL)

type(object)
```

Arguments

`object`	An object.
`tag`	A tag name.
`tags`	A list of tags. • `tag_add`: A named vector with tag names and their values. • `tag_del`: A character vector of tags to delete.

Examples

```
foo <- list()
foo <- tag_add(foo, list(tag_name = "tag_value"))
tag_has(foo, "tag_name") # TRUE
tag_get(foo, "tag_name") # "tag_value"
tag_get(foo, "not_exist") # ""
foo <- tag_del(foo)
tag_has(foo, "tag_name") # FALSE
```

Index