

# Package ‘paws.common’

August 20, 2019

**Type** Package

**Title** Paws Low-Level Amazon Web Services API

**Version** 0.2.0

**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)

**Encoding** UTF-8

**LazyData** true

**Imports** base64enc, digest, httr, ini, jsonlite, methods, utils, xml2

**Suggests** covr, testthat

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

**RoxygenNote** 6.1.1

**Collate** 'struct.R' 'handlers.R' 'url.R' 'net.R'  
'credential\_providers.R' 'credentials.R' 'client.R' 'config.R'  
'convert.R' 'service.R' 'custom\_dynamodb.R' 'custom\_s3.R'  
'dateutil.R' 'error.R' 'handlers\_core.R' 'handlers\_ec2query.R'  
'handlers\_jsonrpc.R' 'handlers\_query.R' 'handlers\_rest.R'  
'handlers\_restjson.R' 'handlers\_restxml.R' 'idempotency.R'  
'jsonutil.R' 'populate.R' 'populateutil.R' 'tags.R'  
'queryutil.R' 'request.R' 'signer\_v4.R' 'signer\_s3.R'  
'signer\_s3v4.R' 'signer\_v2.R' 'time.R' 'util.R' 'xmlutil.R'

**NeedsCompilation** no

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

**Maintainer** David Kretch <[david.kretch@gmail.com](mailto:david.kretch@gmail.com)>

**Repository** CRAN

**Date/Publication** 2019-08-20 05:50:03 UTC

**R topics documented:**

is_empty . . . . .	2
new_handlers . . . . .	3
new_operation . . . . .	3
new_request . . . . .	4
new_service . . . . .	5
populate . . . . .	6
send_request . . . . .	7
tags . . . . .	7

<b>Index</b>	<b>9</b>
--------------	----------

---

is_empty	<i>Check whether an object is empty</i>
----------	---

---

**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
```

---

new_handlers	<i>Return request handlers for a service</i>
--------------	--

---

### 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	<i>Return an API operation object</i>
---------------	---------------------------------------

---

### 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, paginator,
  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.
paginator	Currently unused.
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)
```

## Arguments

client	A service client, e.g. from <code>new_service</code> .
operation	An operation, e.g. from <code>new_operation</code> .
params	A populated input object.
data	An empty output object.

## See Also

Other API request functions: [new\\_handlers](#), [new\\_operation](#), [new\\_service](#), [send\\_request](#)

## Examples

```
# Make a request object for the S3 ListBuckets operation.
client <- function() {new_service(metadata, handlers)}
op <- new_operation("ListBuckets", "GET", "/", list())
params <- list()
data <- tag_add(list(Buckets = list()), list(type = "structure"))
req <- new_request(client, op, params, data)
```

---

new_service	<i>Return an AWS API service object</i>
-------------	---

---

### Description

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

### Usage

```
new_service(metadata, handlers)
```

### 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.

### 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

Other API request functions: [new\\_handlers](#), [new\\_operation](#), [new\\_request](#), [send\\_request](#)

## Examples

```
# 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)
```

---

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)
```

## Arguments

input	A list with data to copy.
interface	A list of a similar shape to copy data into.

## 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	<i>Send a request and handle the response</i>
--------------	---

---

### 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 <code>new_request</code> .
---------	---

### See Also

Other API request functions: [new\\_handlers](#), [new\\_operation](#), [new\\_request](#), [new\\_service](#)

### Examples

```
# Send a request and handle the response.  
resp <- send_request(req)
```

---

tags	<i>Get, set, and delete object tags</i>
------	---

---

### 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_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. <ul style="list-style-type: none"><li>• tag_add: A named vector with tag names and their values.</li><li>• tag_del: A character vector of tags to delete.</li></ul>

**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

`is_empty`, 2

`new_handlers`, 3, 4, 5, 7

`new_operation`, 3, 3, 4, 5, 7

`new_request`, 3, 4, 4, 5, 7

`new_service`, 3, 4, 5, 7

`populate`, 6

`send_request`, 3–5, 7

`tag_add (tags)`, 7

`tag_del (tags)`, 7

`tag_get (tags)`, 7

`tag_has (tags)`, 7

`tags`, 7

`type (tags)`, 7