Package 'restfulr'

June 11, 2026

Type Package

Title R Interface to RESTful Web Services

Version 0.0.17

Description Models a RESTful service as if it were a nested R list.

License Artistic-2.0

Imports XML, RCurl, rjson, S4Vectors (>= 0.13.15), yaml

Depends R (>= 3.4.0), methods

Suggests getPass, rsolr, RUnit

Collate CRUDProtocol-class.R CacheInfo-class.R Credentials-class.R HTTP-class.R Media-class.R MediaCache-class.R RestUri-class.R RestContainer-class.R test_restfulr_package.R utils.R

NeedsCompilation yes

Author Michael Lawrence [aut, cre]

Maintainer Michael Lawrence <lawremi@gmail.com>

Repository CRAN

Date/Publication 2026-06-11 05:10:02 UTC

Contents

Credentials-class *Credentials*

Description

`Credentials` stores authentication credentials (username and password). Note that it is easy to retrieve the password from this object. There is no encryption or other security.

Accessors

- `username`: get the username as a string.
- `password`: get the password as a plain text string.

Author(s)

Michael Lawrence

CRUDProtocol-class *CRUDProtocol*

Description

`CRUDProtocol` is a base class for implementing the communication protocol for performing Create, Read, Update and Delete (CRUD) operations on a resource identified by a `RestUri`. Only HTTP is implemented by this package, and it should serve as useful example for other implementations.

Author(s)

Michael Lawrence

Media-class *Media*

Description

The `Media` object represents a value identified by a URI. There is a `Media` subclass for each media type, such as "text/csv" or "application/xml". Coercion methods (see `setAs`) define mappings between `Media` subclasses and R objects. The user does not usually need to access this functionality directly.

Type conversion

Each Media subclass may be converted to/from one or more R types. The mappings are established by setAs.

The following bi-directional mappings are built into the package:

application/xml, text/html	XMLAbstractNode
application/json	list
text/csv	data.frame
text/*	character

But call mediaCoercionTable to see what is defined in the current session.

The as function is the canonical interface to converting media to R objects. It (usefully) requires that the user specify the target R type. For convenience, the mediaTarget generic returns the default R type for a given Media object.

To support a new media type, one should define a Media subclass with the same name as the media type (application/xml), a corresponding mediaTarget method, and all relevant coerce methods. See the Media class hierarchy to determine where the new type fits.

Helpers

- mediaCoercionTable(): Returns a character matrix with columns "from" and "to", indicating the available coercions of media types to/from R objects.

Author(s)

Michael Lawrence

Examples

```
txt <- '{"json":{"rocks":true}}'
json <- as(txt, "application/json")
as(json, mediaTarget(json))
```

MediaCache-class *MediaCache*

Description

MediaCache is just an environment that restricts the types of its elements to Media. Construct an instance by calling MediaCache. The shared default for all REST clients is returned by globalRestClientCache.

Author(s)

Michael Lawrence

RestContainer-class *RestContainer*

Description

The RestContainer object wraps a collection of resources with a list-like interface. Values are stored and retrieved using familiar accessors like [[and [[<-. Coercion between external media and R objects is based on the Media framework.

Data access

The RestContainer object maps familiar R list accessors to CRUD operations on RestUri.

- x[] <- value: Creates resources for the objects in value at x. This is the create/POST operation. Unlike an R list, the resources are added to the collection without removing any existing resources. This inconsistency is unfortunate, so we might change this behavior in the future.
- x$name, x[[i]]: Reads the named element. This is the read/GET operation.
- x[i]: Reads the named elements, which are returned in a list. This is the vectorized read/GET operation. Unlike an R list, this is not an endomorphism, in that the return value is dropped to a list and is no longer attached to the REST interface.
- x$name <- value, x[[i]] <- value: Updates the named resource with value. This is the update/PUT operation.
- x[i] <- value: Updates resources at x with the objects in value, a list. This is the vectorized update/PUT operation.
- x$name <- NULL, x[[i]] <- NULL: Deletes the named resource. This is the delete/DELETE operation.

Constructor

- RestContainer(...): Constructs an instance based on RestUri(...).

Author(s)

Michael Lawrence

See Also

RestUri, which is a lower-level but perhaps more sensible interface.

Examples

```
apache <- RestContainer("http://wiki.apache.org")
apache$solr
```

RestUri-class *RestUri*

Description

The RestUri object represents a resource accessible via a RESTful interface. It extends character with a protocol, used for accessing the data, as well as a cache. R objects are converted to/from external media via the Media framework.

CRUD interface

There are four canonical, abstract types of REST operations: create, read, update, delete (CRUD). The CRUD model was borrowed from traditional databases. The restfulr package maps those four operations to R functions of the same name. The functions are generic, and there are methods for RestUri and character (taken to be a URI), described below.

- create(x, value, ..., returnResponse=FALSE): Creates a resource at x by converting value to a supported media type. The ... become query parameters on x. If returnResponse is TRUE, convert and return any response sent from the endpoint. By default, x is returned, to support chaining. This corresponds to an HTTP POST.
- read(x, ...): Reads the resource at x, coerces it to an R object, and returns the object. The ... become query parameters on x. This corresponds to an HTTP GET.
- update(object, value, ...): Updates the resource at x by converting value to a supported media type. The ... become query parameters on x. This corresponds to an HTTP PUT.
- delete(x, ...): Deletes the resource at x. This corresponds to an HTTP DELETE.

Constructor

- RestUri(base.uri, protocol = CRUDProtocol(base.uri, ...), cache = globalRestClientCache(), ...): Constructs a RestUri object, pointing to base.uri, a string representation of the URI. The protocol (a CRUDProtocol instance) is automatically determined from the scheme of the URI. By default, all instances share the same global cache, a MediaCache instance.
- x$name: Extends the path of x by appending name. This is a convenient way to narrow a URI and is intuitive if one thinks of a tree of resources as a nested list.
- x[[i]]: Extends the path of x by appending i.
- x[...]: Named arguments in ... become query parameters on x.

Container support

- container(x): Gets a RestContainer object for treating x as a list-like container.

Authentication

RestUri currently supports basic HTTP authentication. Call `authenticate(x)` to add credentials to the RestUri `x`. Retrieve the Credentials object with the `credentials` accessor.

Once a set of credentials has been entered, it is recorded for the URI in '`$(HOME)/.local/config/restfulr/credentials`. The path prefix can be changed via the `XDG_CONFIG_DIR` environment variable, according to the XDG standard. The credential cache is checked during authentication, so that a user does not need to reenter credentials for the same URI.

If the **getPass** package is installed, we use it for password entry. Otherwise, we rely on an implementation that shows the password as it is entered, unless the user is in a terminal, where we can hide input.

Utilities

- `embedCredentials(x)`: Embeds the internal credential information into the URL itself, for interfacing with other tools, like `utils::download.file`.

Author(s)

Michael Lawrence

Examples

```
apache <- RestUri("http://wiki.apache.org")
## Not run:
    read(apache$solr)

## End(Not run)
```

Index