Package 'RCurl'

June 3, 2026

Version 1.98-1.19

Title General Network (HTTP/FTP/...) Client Interface for R

SystemRequirements GNU make, libcurl

Description A wrapper for 'libcurl' <https://curl.se/libcurl/>
Provides functions to allow one to compose general HTTP requests
and provides convenient functions to fetch URIs, get & post
forms, etc. and process the results returned by the Web server.
This provides a great deal of control over the HTTP/FTP/...
connection and the form of the request while providing a
higher-level interface than is available just using R socket
connections. Additionally, the underlying implementation is
robust and extensive, supporting FTP/FTPS/TFTP (uploads and
downloads), SSL/HTTPS, telnet, dict, ldap, and also supports
cookies, redirects, authentication, etc.

License BSD_3_clause + file LICENSE

Depends R (>= 3.4.0), methods

Imports bitops

Suggests XML

Collate aclassesEnums.R bitClasses.R xbits.R base64.R binary.S
classes.S curl.S curlAuthConstants.R curlEnums.R curlError.R
curlInfo.S dynamic.R form.S getFormParams.R getURLContent.R
header.R http.R httpError.R httpErrors.R iconv.R info.S mime.R
multi.S options.S scp.R support.S upload.R urlExists.R zclone.R
zzz.R

BugReports https://codeberg.org/aitap/RCurl

NeedsCompilation yes

Author CRAN Team [ctb] (de facto maintainer in 2013-2026),
Duncan Temple Lang [aut] (ORCID:
<https://orcid.org/0000-0003-0159-1546>),
Ivan Krylov [cre]

Maintainer Ivan Krylov <ikrylov@disroot.org>

Repository CRAN

Date/Publication 2026-06-03 08:50:24 UTC

Contents

AUTH_ANY *Constants for identifying Authentication Schemes*

Description

These variables are symbolic constants that allow use to specify different combinations of schemes for HTTP authentication in a request to a Web server. We can combine them via the | operator to indicate that libcurl should try them in order until one works.

Examples

```
AUTH_BASIC | AUTH_DIGEST
```

base64 *Encode/Decode base64 content*

Description

These functions encode and decode strings using base64 representations. base64 can be used as a single entry point with an argument to encode or decode. The other two functions perform the specific action.

Usage

```
base64(txt, encode = !inherits(txt, "base64"), mode = "character")
```

Arguments

txt	character string to encode or decode
encode	logical value indicating whether the desired action is to encode or decode the object. If txt has (S3) class base64, the default is to decode this.
mode	a character string which is either "raw" or "character". This controls the type of vector that is returned. If this is "raw", a raw vector is created. Otherwise, a character vector of length 1 is returned and its element is the text version of the original data given in txt.

Details

This calls the routines in libcurl. These are not declared in the curl header files. So the support may need to be handled carefully on some platforms, e.g. Microsoft Windows.

Value

If encode is TRUE, a character vector with a class named base64. If decode is TRUE, a simple string.

Note

This is currently not vectorized.

We might extend this to work with raw objects.

Author(s)

Duncan Temple Lang

References

libcurl - https://curl.se/ Wikipedia's explanation of base 64 encoding - https://en.wikipedia.org/wiki/Base64

Examples

```
  # encode and then decode a simple string.
txt = "Some simple text for base 64 to handle"
x = base64(txt)
base64(x)

  # encode to a raw vector
x = base64("Simple text", TRUE, "raw")

  # decode to a character string.
ans = base64Decode(x)
ans == txt
  # decoded to a raw format.
ans = base64Decode(x, "raw")


 # Binary data
#  f = paste(R.home(), "doc", "html", "logo.jpg", sep = .Platform$file.sep)
f = system.file("examples", "logo.jpg", package = "RCurl")
img = readBin(f, "raw", file.info(f)[1, "size"])
b64 = base64Encode(img, "raw")
back = base64Decode(b64, "raw")
identical(img, back)

 # alternatively, we can encode to a string and then decode back again
 # to raw and see that we preserve the date.

enc = base64Encode(img, "character")
dec = base64Decode(enc, "raw")
identical(img, dec)


 # The following would be the sort of computation we could do if we
 # could have in-memory raw connections.
 # We would save() some objects to such an in-memory binary/raw connection
 # and then encode the resulting raw vector into a character vector.
 # Then we can insert that into a message, e.g. an email message or
```

```
 # an XML document and when we receive it in a different R session
 # we would get the string and reverse the encoding from the string to
 # a raw vector
 # In the absence of that in-memory connection  facility in save(),
 # we can use a file.

x = 1:10

 # save two objects - a function and a vector
f = paste(tempfile(), "rda", sep = ".")
save(base64, x, file = f)

 # now read the results back from that file as a raw vector
data = readBin(f, "raw", file.info(f)[1,"size"])

 # base64 encode it
txt = base64Encode(data, "character")

if(require(XML)) {
  tt = xmlTree("r:data", namespaces = c(r = "http://www.r-project.org"))
  tt$addNode(newXMLTextNode(txt))
  out = saveXML(tt)


  doc = xmlRoot(xmlTreeParse(out, asText = TRUE))
  rda = base64Decode(xmlValue(doc), "raw")
  f = tempfile()
  writeBin(rda, f)
  e = new.env()
  load(f, e)
  objects(e)
}

 # we'd like to be able to do
 #  con = rawConnection(raw(), 'r+')
 #  save(base64, x, file = con)
 #  txt = base64Encode(rawConnectionValue(con), "character")
 # ... write and read xml stuff
 #  val = xmlValue(doc)
 #  rda = base64Decode(val, "raw")
 #  e = new.env()
 #  input = rawConnection(o, "r")
 #  load(input, e)
```

---

basicHeaderGatherer *Functions for processing the response header of a libcurl request*

Description

These two functions are used to collect the contents of the header of an HTTP response via the headerfunction option of a curl handle and then processing that text into both the name: value

pairs and also the initial line of the response that provides the status of the request. basicHeaderGatherer is a simple special case of basicTextGatherer with the built-in post-processing step done by parseHTTPHeader.

Usage

```
basicHeaderGatherer(txt = character(), max = NA)
parseHTTPHeader(lines, multi = TRUE)
```

Arguments

txt	any initial text that we want included with the header. This is passed to basicTextGatherer. Generally it should not be specified unless there is a good reason.
max	This is passed directly to basicTextGatherer
lines	the text as a character vector from the response header that parseHTTPHeader will convert to a status and name-value pairs.
multi	a logical value controlling whether we check for multiple HTTP headers in the lines of text. This is caused by a Continue being concatenated with the actual response. When this is TRUE, we look for the lines that start an HTTP header, e.g. HTTP 200 ..., and we use the content from the last of these.

Value

The return value is the same as basicTextGatherer, i.e. a list with update, value and reset function elements. The value element will invoke parseHTTPHeader on the contents read during the processing of the libcurl request and return that value.

Author(s)

Duncan Temple Lang

References

Curl homepage https://curl.se/

See Also

basicTextGatherer curlPerform curlSetOpt

Examples

```
tryCatch(withAutoprint({
   h = basicHeaderGatherer()
   getURI("https://r-project.org",
            headerfunction = h$update)
   names(h$value())
   h$value()
}), GenericCurlError = identity)
```

basicTextGatherer *Cumulate text across callbacks (from an HTTP response)*

Description

These functions create callback functions that can be used to with the libcurl engine when it passes information to us when it is available as part of the HTTP response.

`basicTextGatherer` is a generator function that returns a closure which is used to cumulate text provided in callbacks from the libcurl engine when it reads the response from an HTTP request.

`debugGatherer` can be used with the `debugfunction` libcurl option in a call and the associated `update` function is called whenever libcurl has information about the header, data and general messages about the request.

These functions return a list of functions. Each time one calls `basicTextGatherer` or `debugGatherer`, one gets a new, separate collection of functions. However, each collection of functions (or instance) shares the variables across the functions and across calls. This allows them to store data persistently across the calls without using a global variable. In this way, we can have multiple instances of the collection of functions, with each instance updating its own local state and not interfering with those of the others.

We use an S3 class named `RCurlCallbackFunction` to indicate that the collection of funcions can be used as a callback. The `update` function is the one that is actually used as the callback function in the CURL option. The `value` function can be invoked to get the current state that has been accumulated by the `update` function. This is typically used when the request is complete. One can reuse the same collection of functions across different requests. The information will be cumulated. Sometimes it is convenient to reuse the object but reset the state to its original empty value, as it had been created afresh. The `reset` function in the collection permits this.

`multiTextGatherer` is used when we are downloading multiple URIs concurrently in a single libcurl operation. This merely uses the tools of `basicTextGatherer` applied to each of several URIs. See `getURIAsynchronous`.

Usage

```
basicTextGatherer(txt = character(), max = NA, value = NULL,
                  .mapUnicode = TRUE)
multiTextGatherer(uris, binary = rep(NA, length(uris)))
debugGatherer()
```

Arguments

`txt`	an initial character vector to start things. We allow this to be specified so that one can initialize the content.
`max`	if specified as an integer this controls the total number of characters that will be read. If more are read, the function tells libcurl to stop!
`uris`	for `multiTextGatherer`, this is either the number or the names of the uris being downloaded and for which we need a separate writer function.

value	if specified, a function that is called when retrieving the text usually after the completion of the request and the processing of the response. This function can be used to convert the result into a different format, e.g. parse an XML document, read values from table in the text.
.mapUnicode	a logical value that controls whether the resulting text is processed to map components of the form \uxxxx to their appropriate Unicode representation.
binary	a logical vector that indicates which URIs yield binary content

Details

This is called when the libcurl engine finds sufficient data on the stream from which it is reading the response. It cumulates these bytes and hands them to a C routine in this package which calls the actual gathering function (or a suitable replacement) returned as the update component from this function.

Value

Both the basicTextGatherer and debugGatherer functions return an object of class RCurlCallbackFunction. basicTextGatherer extends this with the class RCurlTextHandler and debugGatherer extends this with the class RCurlDebugHandler. Each of these has the same basic structure, being a list of 3 functions.

update	the function that is called with the text from the callback routine and which processes this text by accumulating it into a vector
value	a function that returns the text cumulated across the callbacks. This takes an argument collapse (and additional ones) that are handed to paste. If the value of collapse is given as NULL, the vector of elements containing the different text for each callback is returned. This is convenient when debugging or if one knows something about the nature of the callbacks, e.g. the regular size that causes iit to identify records in a natural way.
reset	a function that resets the internal state to its original, empty value. This can be used to reuse the same object across requests but to avoid cumulating new input with the material from previous requests.

multiTextGatherer returns a list with an element corresponding to each URI. Each element is an object obtained by calling basicTextGatherer, i.e. a collection of 3 functions with shared state.

Author(s)

Duncan Temple Lang

References

Curl homepage https://curl.se/

See Also

getURL dynCurlReader

Examples

```
tryCatch(withAutoprint({
  txt = getURL("https://r-project.org", write = basicTextGatherer())

  h = basicTextGatherer()
  txt = getURL("https://r-project.org", write = h$update)
    ## Cumulate across pages.
  txt = getURL("https://aitap.grebedoc.dev/RCurl/concurrent.html", write = h$update)


  headers = basicTextGatherer()
  txt = getURL("https://aitap.grebedoc.dev/RCurl/concurrent.html",
               header = TRUE, headerfunction = headers$update)

     ## Now read the headers.
  cat(headers$value())
  headers$reset()


    ## Debugging callback
  d = debugGatherer()
  x = getURL(
    "https://aitap.grebedoc.dev/RCurl/concurrent.html",
    debugfunction = d$update, verbose = TRUE
  )
  cat(names(d$value()))
  d$value()[["headerIn"]]


    ## This hung on Solaris
    ## 2022-02-08 philosophy.html is malformed UTF-8
  uris = c("https://aitap.grebedoc.dev/RCurl/concurrent.html",
           "https://aitap.grebedoc.dev/RCurl/philosophy.html")
## Not run:
  g = multiTextGatherer(uris)
  txt = tryCatch(getURIAsynchronous(uris,  write = g),
                 GenericCurlError = conditionMessage)
  names(txt) # no names this way
  nchar(txt)

   # Now don't use names for the gatherer elements.
  g = multiTextGatherer(length(uris))
  txt = tryCatch(getURIAsynchronous(uris,  write = g),
                 GenericCurlError = conditionMessage)
  names(txt)
  nchar(txt)

## End(Not run)
}),
GenericCurlError = identity)
```

```
## Not run:
 Sys.setlocale(,"en_US.latin1")
 Sys.setlocale(,"en_US.UTF-8")
 uris = c("https://aitap.grebedoc.dev/RCurl/concurrent.html",
          "https://aitap.grebedoc.dev/RCurl/philosophy.html")
 g = multiTextGatherer(uris)
 txt = getURIAsynchronous(uris,  write = g)

## End(Not run)
```

`binaryBuffer` *Create internal C-level data structure for collecting binary data*

Description

This is the constructor function for creating an internal data structure that is used when reading binary data from an HTTP request via RCurl. It is used with the native routine `R_curl_write_binary_data` for collecting the response from the HTTP query into a buffer that stores the bytes. The contents can then be brought back into R as a `raw` vector and then used in different ways, e.g. uncompressed with the `Rcompression` package, or written to a file via `writeBin`.

Usage

```
binaryBuffer(initialSize = 5000)
```

Arguments

`initialSize`	a number giving the size (number of bytes) to allocate for the buffer. In most cases, the size won't make an enormous difference. If this is small, the `R_curl_write_binary_data` routine will expand it as necessary when more daat is received than would fit in it. If it is very large, i.e. larger than the resulting response, the consequence is simply unused memory. One can determine the appropriate size by performing the HTTP request with `nobody = TRUE` and looking at the resulting size indicated by the headers of the response, i.e. `getCurlInfo(handle)` and then using that size and repeating the request and receiving the body. This is a trade-off between network speed and memor consumption and processing speed when collecting the .

Value

An object of class `RCurlBinaryBuffer` which is to be treated as an opaque data for the most part. When passing this as the value of the `file` option, one will have to pass the ref slot.

After the contents have been read, one can convert this object to an R raw vector using `as(buf, "raw")`.

Author(s)

Duncan Temple Lang

References

Curl homepage https://curl.se/

See Also

R_curl_write_binary_data

Examples

```
tryCatch(withAutoprint({
  buf = binaryBuffer()

     # Now fetch the binary file.
  getURI("https://aitap.grebedoc.dev/RCurl/xmlParse.html.gz",
         write = getNativeSymbolInfo("R_curl_write_binary_data")$address,
         file = buf@ref)

   # Convert the internal data structure into an R raw vector
  b = as(buf, "raw")

  if (getRversion() >= "4")
    txt = memDecompress(b, asChar = TRUE)
  ## or txt = Rcompression::gunzip(b)
}), GenericCurlError = identity)
```

CFILE *Create a C-level handle for a file*

Description

This function and class allow us to work with C-level FILE handles. The intent is to be able to pass these to libcurl as options so that it can read or write from or to the file. We can also do this with R connections and specify callback functions that manipulate these connections. But using the C-level FILE handle is likely to be significantly faster for large files.

The close method allows us to explicitly flush and close the file from within R.

Usage

```
CFILE(filename, mode = "r")
```

Arguments

filename	the name of the file on disk
mode	a string specifying how to open the file, read or write, text or binary.

Details

This is a simple interface to the C routine fopen.

Value

An object of class CFILE which is has a single slot name ref which is an external pointer holding the address of the FILE object in C.

Author(s)

Duncan Temple Lang

References

Man page for fopen

See Also

curlPerform and the readdata

Examples

```
## Not run:
filename = system.file("tests", "amazon3.R", package = "RCurl")
f = CFILE(filename)

if(url.exists('http://s3.amazonaws.com/'))
  curlPerform(url = "http://s3.amazonaws.com/RRupload/duncan2",
              upload = TRUE,
              readdata = f@ref,
              infilesize = file.info(filename)[1, "size"])

## End(Not run)
```

---

chunkToLineReader *Utility that collects data from the HTTP reply into lines and calls user-provided function.*

Description

When one provides an R function to process the body of the R rep

Usage

```
chunkToLineReader(f, verbose = FALSE)
```

Arguments

f	a function that is to be called each time the read function is invoked and there are complete lines in that input.
verbose	a logical value. If TRUE, information is displayed when there is any text that does not form a complete line and is held for processing in the next chunk.

Details

This constructs a closure and then processes each chunk as they are passed to the read function. It strips away any text that does not form a complete line at the end of the chunk and holds this to be added to the next chunk being processed.

Value

A list with two components

`read`	the function that will do the actual reading from the HTTP response stream and call the function `f` on each step (assuming the chunk has a line marker.
`comp2`	Description of 'comp2'

...

Author(s)

Duncan Temple Lang

References

Curl homepage https://curl.se/

See Also

`getURI` and the `write` argument. `getForm`, `postForm` `curlPerform`

Examples

```
# Read a rectangular table of data into R from the URL
# and add up the values and the number of values read.

summer =
function()
{
  total = 0.0
  numValues = 0

  list(read = function(txt) {
                con = textConnection(txt)
                on.exit(close(con))
                els = scan(con)
                numValues <<- numValues + length(els)
                total <<- total + sum(els)

                ""
              },
       result = function() c(total = total, numValues = numValues))
}

s = summer()
```

```
tryCatch(
 getURL("https://aitap.grebedoc.dev/RCurl/matrix.data", write = chunkToLineReader(s$read)$read),
  GenericCurlError = identity
)
```

---

clone *Clone/duplicate an object*

Description

This is a generic function and methods for making a copy of an object such as a curl handle, C-level pointer to a file, etc.

Usage

```
clone(x, ...)
```

Arguments

x	the object to be cloned.
...	additional parameters for methods

Value

Typically, an object of the same class and "value" as the input - x.

Author(s)

Duncan Temple Lang

See Also

dupCurlHandle

Examples

```
h = getCurlHandle(verbose = TRUE)
other = dupCurlHandle(h)
curlSetOpt(curl = h, verbose = FALSE)
```

complete *Complete an asynchronous HTTP request*

Description

This is a generic function that is used within the RCurl package to force the completion of an HTTP request. If the request is asynchronous, this essentially blocks until the request is completed by repeatedly asking for more information to be retrieved from the HTTP connection.

Usage

```
complete(obj, ...)
```

Arguments

obj	the object which is to be completed. This is typically a MultiCURLHandle-class instance.
...	additional arguments intended to be used by specific methods.

Value

The value is typically not of interest, but rather the side effect of processing the pending requests.

Author(s)

Duncan Temple Lang

References

https://curl.se/, specifically the multi interface of libcurl.

See Also

MultiCURLHandle-class push, pop

Examples

```
## Not run: # it does not exist
if(url.exists("http://eeyore.ucdavis.edu/cgi-bin/testForm1.pl")) {

  f = system.file("NAMESPACE", package = "RCurl")
   postForm("http://eeyore.ucdavis.edu/cgi-bin/testForm1.pl",
            "fileData" = fileUpload(f))


   postForm("http://eeyore.ucdavis.edu/cgi-bin/testForm1.pl",
            "fileData" = fileUpload("",
                                    paste(readLines(f), collapse = "\n"),
                                    "text/plain"))
```

```
   postForm("http://eeyore.ucdavis.edu/cgi-bin/testForm1.pl",
            "fileData" = fileUpload(f,
                                    paste(readLines(f), collapse = "\n")
                                    ),
            .opts = list(verbose = TRUE, header = TRUE))
}
## End(Not run)
```

CURLEnums *Classes and coercion methods for enumerations in libcurl*

Description

These are classes and coercion methods for enumeration types in RCurl corresponding to symbolic constants in libcurl. The actual constants are not exported, but are defined within the package. So we can use them with code such as `RCurl:::CURLINFO_DATA_IN`.

Author(s)

Duncan Temple Lang

---

curlError *Raise a warning or error about a CURL problem*

Description

This function is called to raise an error or warning that arises from a curl operation when making a request. This is called from C code that encounters the error and this function is responsible for generating the error.

Usage

```
curlError(type, msg, asError = TRUE)
```

Arguments

type	the type of the error or a status code identifying the type of the error. Typically this is an integer value that identifies the type of the Curl error. The value corresponds to one of the enumerated value of type CURLcode.
msg	the error message, as a character vector of length 1
asError	a logical value that indicates whether to raise an error or a warning

Value

This calls `warning` or `stop` with the relevant condition object. The object is always of basic (S3) class `GenericCurlError`, `error`, `condition` or `GenericCurlError`, `warning`, `condition`. When the `type` value corresponds to a `CURLCode` value, the condition has the primary class given by that `CURLCode`'s name, e.g. `COULDNT_RESOLVE_HOST`, `TOO_MANY_REDIRECTS` (with the CURLE prefix removed).

Author(s)

Duncan Temple Lang

References

libcurl documentation.

See Also

`curlPerform`

Examples

```
 # This illustrates generating and catching an error.
 # We intentionally give a mis-spelled URL.
tryCatch(curlPerform(url = "ftp.wcc.nrcs.usda.govx"),
         COULDNT_RESOLVE_HOST = function(x) cat("resolve problem\n"),
         error = function(x) cat(class(x), "got it\n"))
```

---

curlEscape *Handle characters in URL that need to be escaped*

Description

These functions convert between URLs that are human-readable and those that have special characters escaped. For example, to send a URL with a space, we need to represent the space as `%20`.

`curlPercentEncode` uses a different format than the `curlEscape` function and this is needed for x-www-form-encoded POST submissions.

Usage

```
curlEscape(urls)
curlUnescape(urls)
curlPercentEncode(x, amp = TRUE, codes = PercentCodes, post.amp = FALSE)
```

Arguments

urls	a character vector giving the strings to be escaped or unescaped.
x	the strings to be encoded via the percent-encoding method
amp	a logical value indicating whether to encode & characters.
codes	the named character vector giving the encoding map. The names are the characters we encode, the values are what we encode them as.
post.amp	a logical value controlling whether the resulting string is further processed to escape the percent (%) prefixes with the code for percent, i.e. %25.

Details

This calls curl_escape or curl_unescape in the libcurl library.

Value

A character vector that has corresponding elements to the input with the characters escaped or not.

Author(s)

Duncan Temple Lang

References

Curl homepage https://curl.se/

Percent encoding explained in https://en.wikipedia.org/wiki/Percent-encoding

Examples

```
curlEscape("http://www.abc.com?x=a is a sentence&a b=and another")

# Reverse it should get back original
curlUnescape(curlEscape("http://www.abc.com?x=a is a sentence&a b=and another"))
```

CurlFeatureBits *Constants for libcurl*

Description

These are enums and bit fields defining constants used in libcurl and used in R to specify values symbolically.

Usage

```
CurlFeatureBits
```

Format

named integer vectors. The names give the symbolic constants that can be used in R and C code. These are mapped to their integer equivalents and used in C-level computations.

Source

libcurl (see https://curl.se/)

`curlGlobalInit` *Start and stop the Curl library*

Description

These functions provide a way to both start/initialize and stop/uninitialize the libcurl engine. There is no need to call `curlGlobalInit` as it is done implicitly the first time one uses the libcurl facilities. However, this function does permit one to explicitly control how it is initialized. Specifically, on Windows one might want to avoid re-initializing the Win32 socket facilities if the host application (e.g. R) has already done this.

`curlGlobalInit` should only be called once per R session. Subsequent calls will have no effect, or may confuse the libcurl engine.

One can reclaim the resources the libcurl engine is consuming via the `curlGlobalCleanup` function when one no longer needs the libcurl facilities in an R session.

Usage

```
curlGlobalInit(flags = c("ssl", "win32"))
curlGlobalCleanup()
```

Arguments

`flags`	flags indicating which features to activate. These come from the `CurlGlobalBits` bit-field. The default is to activate both SSL and Win32 sockets (if on Windows). One can specify either the names of the features that are matched (via `setBitIndicators`) or the integer value.

Value

`curlGobalInit` returns a status code which should be 0. `curlGlobalCleanup` returns `NULL` in all cases.

Author(s)

Duncan Temple Lang

References

Curl homepage https://curl.se/

See Also

getCurlHandle curlPerform

Examples

```
    # Activate only the SSL.
 curlGlobalInit("ssl")

## Not run:
# Don't run these automatically as we should only call this function
# once per R session

    # Everything, the default.
 curlGlobalInit()

   # Nothing.
 curlGlobalInit("none")
 curlGlobalInit(0)

## End(Not run)
```

CURLHandle-class *Class "CURLHandle" for synchronous HTTP requests*

Description

This is the basic class used for performing HTTP requests in the RCurl package. In R, this is a reference to a C-level data structure so we treat it as an opaque data type. However, essentially we can think of this as an with a set of options that persists across calls, i.e. HTTP requests. The numerous options that one can set can be see via getCurlOptionsConstants. The object can keep a connection to a Web server open for a period of time across calls.

This class differs from MultiCURLHandle-class as it can handle only one request at a time and blocks until the request is completed (normally or abnormally). The other class can handle asynchronous, multiple connections.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "oldClass", directly.

Author(s)

Duncan Temple Lang

References

https://curl.se/, the libcurl web site.

See Also

getURL, getForm, postForm dupCurlHandle, getCurlHandle, MultiCURLHandle-class

---

curlOptions *Constructor and accessors for CURLOptions objects*

Description

These functions provide a constructor and accessor methods for the (currently S3) class CURLOptions. This class is a way to group and manage options settings for CURL. These functions manage a named list of options where the names are elements of a fixed. Not all elements need be set, but these functions take care of expanding names to match the fixed set, while allowing callers to use abbreviated/partial names. Names that do not match (via pmatch) will cause an error.

The set of possible names is given by names(getCurlOptionsConstants()) or more directly with listCurlOptions().

mapCurlOptNames handles the partial matching and expansion of the names of the options for all the functions that handle CURL options. Currently this uses pmatch to perform the matching and so rejects words that are ambiguous, i.e. have multiple matches within the set of permissible option names. As a result, "head" will match both "header" and "headerfunction". We may change this behavior in the future, but we encourage using the full names for readability of code if nothing else.

Usage

```
curlOptions(..., .opts = list())
getCurlOptionsConstants()
## S3 replacement method for class 'CURLOptions'
x[i] <- value
## S3 replacement method for class 'CURLOptions'
x[[i]] <- value
listCurlOptions()
getCurlOptionTypes(opts = getCurlOptionsConstants())
```

Arguments

...	name-value pairs identifying the settings for the options of interest.
.opts	a named list of options, typically a previously created CURLOptions object. These are merged with the options specified in ....
x	a CURLOptions object
i	the name(s) of the option elements being accessed. These can be partial names matching elements in the set of known options. Other names will cause an error.
value	the values to assign to the options identified via i.
opts	the options whose type description are of interest in the call.

Details

These functions use `mapCurlOptNames` to match and hence expand the names the callers provide.

Value

`curlOptions` returns an object of class `CURLOptions` which is simply a named list.

`getCurlConstants` returns a named vector identifying the names of the possible options and their associated values. These values are used in the C code and also each integer encodes the type of the argument expected by the C code for that option.

`getCurlOptionTypes` returns human-readable, heuristic descriptions of the types expected for the different options. These are integer/logical corresponding to "long" in the RCurl documentation; string/object pointer corresponding to "char *" or ; function corresponding to a function/routine pointer; large number corresponding to a `curl_off_t`.

Author(s)

Duncan Temple Lang

References

Curl homepage https://curl.se/

See Also

`curlPerform` `curlSetOpt`

Examples

```
tt = basicTextGatherer()
myOpts = curlOptions(verbose = TRUE, header = TRUE, writefunc = tt[[1]])

 # note that the names are expanded, e.g. writefunc is now writefunction.
names(myOpts)

myOpts["header"]]

myOpts["header"]] <- FALSE

# Using the abbreviation "hea" is an error as it matches
# both
#  myOpts["hea"]] <- FALSE

 # Remove the option from the list
myOpts["header"]] <- NULL
```

curlPerform *Perform the HTTP query*

Description

These function causes the HTTP query, that has been specified via the different options in this and other calls, to be sent and processed. Unlike in curl itself, for curlPerform one can specify all the options in this call as an atomic invocation. This avoids having to set the options and then perform the action. Instead, this is all done in one call.

For curlMultiPerform, one must add the relevant CURLHandle-class objects to the MultiCURLHandle-class objects before issuing the call to curlMultiPerform.

Usage

```
curlPerform(..., .opts = list(), curl = getCurlHandle(), .encoding = integer())
curlMultiPerform(curl, multiple = TRUE)
```

Arguments

curl	for curlPerform, this is the CURLHandle object giving the structure for the options and that will process the command. For curlMultiPerform, this is an object of class code MultiCURLHandle-class.
...	a named list of curl options to set after the handle has been created.
.opts	a named list or CURLOptions object identifying the curl options for the handle. This is merged with the values of ...to create the actual options for the curl handle in the request.
multiple	a logical value. If TRUE and the internal call to curl_multi_perform returns a value that indicates there is still data available from one of the HTTP responses, we call curl_multi_perform repeatedly until there is no more data available at that time. If this is FALSE, we call curl_multi_perform once and return, regardless of whether there is more data available. This is convenient if we want to limit the time spent in the call to curlMultiPerform.
.encoding	an integer or a string that explicitly identifies the encoding of the content that is returned by the HTTP server in its response to our query. The possible strings are 'UTF-8' or 'ISO-8859-1' and the integers should be specified symbolically as CE_UTF8 and CE_LATIN1. Note that, by default, the package attempts to process the header of the HTTP response to determine the encoding. This argument is used when such information is erroneous and the caller knows the correct encoding. Note that the encoding argument is not a regular libcurl option and is handled specially by RCurl. But as a result, it is not unset in subsequent uses of the curl handle (curl).

Value

A integer value indicating the status of the request. This should be 0 as other errors will generate errors.

Author(s)

Duncan Temple Lang

References

Curl homepage https://curl.se/

See Also

getURL postForm getForm curlSetOpt

Examples

```
tryCatch(withAutoprint({
  h = basicTextGatherer()
  curlPerform(url = "https://r-project.org", writefunction = h$update)
   # Now read the text that was cumulated during the query response.
  cat(h$value())
}), GenericCurlError = identity)


## Not run:
## this no longer exists
if(url.exists("http://services.soaplite.com/hibye.cgi")) withAutoprint({
     # SOAP request
  body = '<?xml version="1.0" encoding="UTF-8"?>\
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" \
                   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" \
                   xmlns:xsd="http://www.w3.org/1999/XMLSchema" \
                   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" \
                   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">\
  <SOAP-ENV:Body>\
       <namesp1:hi xmlns:namesp1="http://www.soaplite.com/Demo"/>\
  </SOAP-ENV:Body>\
</SOAP-ENV:Envelope>\n'


  h$reset()
  curlPerform(url = "http://services.soaplite.com/hibye.cgi",
              httpheader=c(Accept="text/xml", Accept="multipart/*",
                           SOAPAction='"http://www.soaplite.com/Demo#hi"',
                           'Content-Type' = "text/xml; charset=utf-8"),
              postfields=body,
              writefunction = h$update,
              verbose = TRUE
             )

  body = h$value()

})

## End(Not run)
```

```
   # Using a C routine as the reader of the body of the response.
tryCatch(withAutoprint({
  routine = getNativeSymbolInfo("R_internalWriteTest", PACKAGE = "RCurl")$address
  curlPerform(URL = "https://r-project.org",
              writefunction = routine)
}), GenericCurlError = identity)
```

---

curlSetOpt *Set values for the CURL options*

Description

This function allows us to set values for the possible options in the CURL data structure that defines the HTTP request. These options persist across calls in the CURLHandle object.

Usage

```
curlSetOpt(..., .opts = list(), curl = getCurlHandle(),
            .encoding = integer(), .forceHeaderNames = FALSE,
             .isProtected = FALSE)
```

Arguments

...	a named list of curl options to set after the handle has been created.
.opts	a named list or CURLOptions object identifying the curl options for the handle.
curl	the CURLHandle object created earlier via a call to getCurlHandle or dupCurlHandle
.encoding	an integer or a string that explicitly identifies the encoding of the content that is returned by the HTTP server in its response to our query. The possible strings are 'UTF-8' or 'ISO-8859-1' and the integers should be specified symbolically as CE_UTF8 and CE_LATIN1. Note that, by default, the package attempts to process the header of the HTTP response to determine the encoding. This argument is used when such information is erroneous and the caller knows the correct encoding.
.forceHeaderNames	a logical value which if TRUE allows the caller to explicitly indicate that the HTTPHEADER option needs to have the names prefixed to the strings. This removes any ambiguity caused by the presence of ':' in the values appearing to be the separator between the name and the value of the name: value pairs of the HTTP header.
.isProtected	a logical vector (or value that is repeated) specifying which of the values in ... and .opts need to be explicitly protected from garbage collection or not. The basic idea is that we specify FALSE if the value being set for the curl option may be garbage collected before the curl handle is garbage collected. This would leave the curl object in an inconsistent state, referring to an R object (i.e. an R function), which may be used after the R object has been garbage collected.

Value

An integer value giving the status of the return. This should be 0 as if there was an error in the libcurl mechiansim, we will throw it there.

Author(s)

Duncan Temple Lang

References

Curl homepage https://curl.se/

See Also

getCurlHandle dupCurlHandle

Examples

```
tryCatch({

  curl = getCurlHandle()
     # Note the header that extends across two lines with the second line
     # prefixed with white space.
   curlSetOpt( .opts = list(httpheader = c(Date = "Wed, 1/2/2000 10:01:01",
                              foo="abc\n    extra line"), verbose = TRUE),
               curl = curl)
   ans = getURL("https://r-project.org", curl = curl)
}, GenericCurlError = identity)
```

---

curlVersion *Information describing the Curl library*

Description

This function queries the Curl library to provide information about its characteristics when it was compiled. This tells the user about its capabilities and can be used to determine strategies.

Usage

```
curlVersion(id = 0)
```

Arguments

id	an integer value between 0 and 3 inclusive. The idea is that one specifies the identifier for the version of interest. In fact, all values seem to yield the same result.

Value

A list

age	integer giving the number of this libcurl, 0 is FIRST, 1 is SECOND, 2 is THIRD
version	the version identifier as a string, e.g. "7.12.0"
version_num	the value as an integer
host	the machine on which the libcurl was configured/built.
features	a named integer vector of bits indicating what features of libcurl were configured and built into this version. These are features such as ipv6, ssl, libz, largefile, ntlm (Microsoft "authorization").
ssl_version	the string identifying the SSL version.
ssl_version_num	the number identifying the SSL version
libz_version	the string identifying the version of libz.
protocols	a character vector of the supported HTTP protocols, e.g. http, https, ftp, ldap, gopher, telnet
ares	name of the asynchronous DNS (domain name service) lookup library. This is often simply the empty string indicating it is not there.
ares_num	the number for the ares library
libidn	the name of the IDN (internationalized domain names) library being used. This field only appears in version 3 of libcurl. If you are using version 2 (e.g. curl-7.11.2), this will be NA. An empty string indicates that the field is present, but has no value.

See the man page for curl_version_info for a description of these fields. features in R is a named integer vector detailing the different features.

Author(s)

Duncan Temple Lang

References

Curl homepage https://curl.se/

See Also

curl_version_info in the libcurl documentation.

Examples

```
curlVersion()
```

`dynCurlReader` *Dynamically determine content-type of body from HTTP header and set body reader*

Description

This function is used for the `writefunction` option in a curl HTTP request. The idea is that we read the header of the HTTP response and when our code determines that the header is complete (the presence of a blank line), it examines the contents of the header and finds a Content-Type field. It uses the value of this to determine the nature of the body of the HTTP response and dynamically (re)sets the reader for the curl handle appropriately. If the content is binary, it collects the content into a `raw` vector; if it is text, it sets the appropriate character encoding and collects the content into a character vector.

This function is like `basicTextGatherer` but behaves dynamically by determining how to read the content based on the header of the HTTP response. This function returns a list of functions that are used to update and query a shared state across calls.

Usage

```
dynCurlReader(curl = getCurlHandle(), txt = character(), max = NA,
              value = NULL, verbose = FALSE, binary = NA, baseURL = NA,
              isHTTP = NA, encoding = NA)
```

Arguments

`curl`	the curl handle to be used for the request. It is essential that this handle be used in the low-level call to `curlPerform` so that the update element sets the reader for the body on the appropriate curl handle that is used in the request.
`txt`	initial value of the text. This is almost always an empty character vector.
`max`	the maximum number of characters to read. This is almost always `NA`.
`value`	a function that can be specified which will be used to convert the body of the response from text or raw in a customized manner, e.g. uncompress a gzip body. This can als be done explicitly with a call `fun(reader$value())` after the body has been read. The advantage of specifying the function in the constructor of the reader is that the end-user doesn't have to know which function to use to do the conversion.
`verbose`	a logical value indicating whether messages about progress and operations are written on the console as the header and body are processed.
`binary`	a logical value indicating whether the caller knows whether the resulting content is binary (`TRUE`) or not (`FALSE`) or unknown (`NA`).
`baseURL`	the URL of the request which can be used to follow links to other URLs that are described relative to this.
`isHTTP`	a logical value indicating whether the request/download uses HTTP or not. If this is `NA`, we determine this when the header is received. If the caller knows this is an FTP or other request, they can specify this when creating the reader.

encoding — a string that allows the caller to specify and override the encoding of the result. This is used to convert text returned from the server.

Value

A list with 5 elements all of which are functions. These are

update — the function that does the actual reading/processing of the content that libcurl passes to it from the header and the body. This is the work-horse of the reader.

value — a function to get the body of the response

header — a function to get the content of the HTPP header

reset — a function to reset the internal contents which allows the same reader to be re-used in subsequent HTTP requests

curl — accessor function for the curl handle specified in the call to create this dynamic reader object.

This list has the S3 class vector `c("DynamicRCurlTextHandler", "RCurlTextHandler", "RCurlCallbackFunction")`

Author(s)

Duncan Temple Lang

References

libcurl https://curl.se/

See Also

basicTextGatherer curlPerform getURLContent

Examples

```
   # Each of these examples can be done with getURLContent().
   # These are here just to illustrate the dynamic reader.
tryCatch(withAutoprint({
  header = dynCurlReader()
  curlPerform(url = "https://www.r-project.org/Rlogo.png",
              headerfunction = header$update, curl = header$curl())
  class( header$value() )
  length( header$value() )
}), GenericCurlError = identity)

tryCatch(withAutoprint({
     # gzip example.
  header = dynCurlReader()
  curlPerform(url = "https://aitap.grebedoc.dev/RCurl/dd.gz",
              headerfunction = header$update, curl = header$curl())
  class( header$value() )
  length( header$value() )

  if (getRversion() >= "4")
```

```
     cat(memDecompress(header$value(), asChar = TRUE))
   ## or   cat(Rcompression::gunzip(header$value()))
}), GenericCurlError = identity)


   # Character encoding example
## Not run:
  header = dynCurlReader()
  curlPerform(url = "http://www.razorvine.net/test/utf8form/formaccepter.sn",
              postfields = c(text = "ABC", outputencoding =  "UTF-8"),
              verbose = TRUE,
              writefunction = header$update, curl = header$curl())
  class( header$value() )
  Encoding( header$value() )

## End(Not run)
```

`fileUpload` *Specify information about a file to upload in an HTTP request*

Description

This function creates an object that describes all of the details needed to include the contents of a file in the submission of an HTTP request, typically a multi-part form submitted via `postForm`. The idea is that we want to transfer the contents of a file or a buffer of data within R that is not actually stored on the file system but is resident in the R session. We want to be able to specify either the name of the file and have RCurl read the contents when they are needed, or alternatively specify the contents ourselves if it makes sense that we already have the contents in R, e.g. that they are dynamically generated. Additionally, we may need to specify the type of data in the file/buffer via the Content-Type field for this parameter in the request. This function allows us to specify either the file name or contents and optionally the content type.

This is used as an element in of the `params` argument `postForm` and the native C code understands and processes objects returned from this function.

Usage

```
fileUpload(filename = character(), contents = character(), contentType = character())
```

Arguments

`filename`	the name of the file that RCurl is to pass in the form submission/HTTP request. If this is specified and no value for `contents` is given, this has to identify a valid/existing file. If `contents` is specified, any value provided here is used simply to provide information about the provenance of the data in contents. The file need not exist. The path is expanded by the function, so ~ can be used.
`contents`	either a character vector or a `raw` vector giving the contents or data to be submitted. If this is provided, `filename` is not needed and not read.

contentType	a character string (vector of length 1) giving the type of the content, e.g. text/plain, text/html, which helps the server receiving the data to interpret the contents. If omitted, this is omitted from the form submission and the recipient left to guess.

Value

An object of (S3) class `FileUploadInfo` with fields `filename`, `contents` and `contentType`.

Author(s)

Duncan Temple Lang

References

https://curl.se/

See Also

postForm

---

findHTTPHeaderEncoding

Find the encoding of the HTTP response from the HTTP header

Description

This function is currently made available so it can be called from C code to find the charset from the HTTP header in the response from an HTTP server. It maps this charset value to one of the known R encodings (UTF-8 or LATIN1) or returns the native encoding.

This will most likely be removed in the future.

Usage

```
findHTTPHeaderEncoding(str)
```

Arguments

str	one or more lines from the HTTP header

Value

`NA` or an integer value indicating the encoding to be used. This integer corresponds to the `cetype_t` enumeration in Rinternals.h.

Author(s)

Duncan Temple Lang

References

Writing R Extensions Manual and the section(s) on character encodings

Examples

```
findHTTPHeaderEncoding("Content-Type: text/html;charset=ISO-8859-1\r\n")
findHTTPHeaderEncoding("Content-Type: text/html; charset=utf-8\r\n")
```

---

ftpUpload *Upload content via FTP*

Description

This function is a relatively simple wrapper for curlPerform which allows the caller to upload a file to an FTP server. One can upload the contents of a file from the local file system or the contents already in memory. One specifies the FTP server and the fully-qualified file name and path where the contents are to be stored. One can specify the user login and password via the userpwd option for curlPerform via the ... parameter, or one can put this information directly in the target URL (i.e. to) in the form ftp://login:password@machine.name/path/to/file.

This function can handle binary or text content.

Usage

```
ftpUpload(what, to, asText = inherits(what, "AsIs") || is.raw(what),
          ..., curl = getCurlHandle())
```

Arguments

what	the name of a local file or the contents to be uploaded. This can can be text or binary content. This can also be an open connection. If this value is raw or has class AsIs by being enclosed within I(), it is treated as literal content.
to	the URL to which the content is to be uploaded. This should be the ftp server with the prefix ftp:// and optionally the user login and password, and then the path to the file in which the content is to be stored.
asText	a logical value indicating whether to treat the value of what as content, be it text or raw/binary vector. Otherwise, what is treated as the name of a file.
...	additional arguments passed on to curlPerform.
curl	the curl handle to use for the curlPerform

Value

The result of the curlPerform call.

Note

One can also provide additional FTP commands that are executed before and after the upload as part of the request. Use the prequote, quote, and postquote options in curlPerform for these.

Author(s)

Duncan Temple Lang

References

FTP, libcurl

See Also

curlPerform getCurlHandle

Examples

```
 ## Not run:

  ftpUpload(I("Some text to be uploaded into a file\nwith several lines"),
            "ftp://login:password@laptop17/ftp/zoe",
            )

  ftpUpload(I("Some text to be uploaded into a file\nwith several lines"),
            "ftp://laptop17/ftp/zoe",
            userpwd = "login:password"
            )

  ftpUpload(system.file("examples", "system.png", package = "RCurl"),
            "ftp://login:password@laptop17/ftp/Election.rda",
            postquote = c("CWD subdir", "RNFR Election.rda", "RNTO ElectionPolls.rda")
            )

## End(Not run)
```

---

getBinaryURL *Download binary content*

Description

This function allows one to download binary content. This is a convenience function that is a call to getURL with suitable values for the `write` and `file` options for the Curl handle. These take care of processing the body of the response to the Curl request into a vector of "raw" elements.

Binary content from POST forms or other requests that are not simple URL requests can be implemented using the same approach as this function, i.e., specifying the same values as in the body of this function for `write` and `file` in the call to curlPerform.

Usage

```
getBinaryURL(url, ..., .opts = list(), curl = getCurlHandle(),
             .buf = binaryBuffer(.len), .len = 5000)
```

Arguments

url	the URL identifying the content to download. This can be a regular URL or a application/x-www-form-urlencoded URL, i.e. with name=value parameters appended to the location via a ?, and separated from each other via a &.
...	additional arguments that are passed to getURL.
.opts	a list of named values that are passed to getURL as the .opts argument.
curl	an optional curl handle used in curlPerform that has been created previously and is to be reused for this request. This allows the R user to reuse a curl handle that already has a connection to the server or has settings for options that have been set previously.
.buf	a raw vector in which to insert the body of the response. This is a parameter to allow the caller to reuse an existing buffer.
.len	an non-negative integer which is used as the length for the buffer in which to store the binary data in the response. The buffer is extended if it is not big enough but this allows the caller to provide context specific knowledge about the length of the response, e.g. the size of the file being downloaded, and avoid expanding the buffer as the material is being processed.

Value

A "raw" vector.

Author(s)

Duncan Temple Lang

See Also

getURL, raw, memDecompress

Examples

```
  u = "https://aitap.grebedoc.dev/RCurl/data.gz"

tryCatch(withAutoprint({

  content = getBinaryURL(u)

  if (getRversion() >= "4") withAutoprint({
    x <- memDecompress(content, asChar = TRUE)
    read.csv(textConnection(x))
  }) else withAutoprint({
     tmp = tempfile()
     writeBin(content, con = tmp)
     read.csv(gzfile(tmp))
     unlink(tmp)
  })
```

```
 # Working from the Content-Type in the header of the HTTP response.
h  = basicTextGatherer()
content = getBinaryURL(u, .opts = list(headerfunction = h$update))
header = parseHTTPHeader(h$value())
type = strsplit(header["Content-Type"], "/")[[1]]

if(type[2] %in% c("x-gzip", "gzip")) {
if (getRversion() >= "4") {
   cat(memDecompress(content, asChar = TRUE))
} else {
  tmp = tempfile()
  writeBin(content, con = tmp)
  writeLines(readLines(gzfile(tmp)))
  unlink(tmp)
  }
}

}), GenericCurlError = identity)
```

`getBitIndicators` *Operate on bit fields*

Description

The `getBitIndicators` function decompose a value into its respective bit components. The `setBitIndicators` combines individual components into a single number to "set" a bit field value.

Usage

```
getBitIndicators(val, defs)
setBitIndicators(vals, defs)
```

Arguments

`val`	the value to break up into the bit field components.
`defs`	the named integer vector that defines the bit field elements.
`vals`	the individual components that are to be combined into a single integer value representing the collection of components. These can be given as names or integer values that correspond to the elements in the defs, either by name or value.

Value

`getBitIndicators` returns a named integer vector representing the components of the bit field in the value. The names of the vector give the symbolic elements that were set in the value.

`setBitIndicators` returns a single integer value representing the value from combining the different components (e.g. ORing the bits of the different values).

Author(s)

Duncan Temple Lang

References

Curl homepage https://curl.se/

See Also

The features field in curlVersion.

Examples

```
getBitIndicators(7, c(A = 1, B = 2, C = 4))
getBitIndicators(3, c(A = 1, B = 2, C = 4))
getBitIndicators(5, c(A = 1, B = 2, C = 4))
```

---

getCurlErrorClassNames

Retrieve names of all curl error classes

Description

This function returns the names of all of the error classes that curl can raise as a result of a request. You can use these names in calls to tryCatch to identify the class of the error for which you want to provide an error handler.

Usage

```
getCurlErrorClassNames()
```

Value

A character vector

Author(s)

Duncan Temple Lang

References

libcurl documentation

See Also

tryCatch curlPerform and higher-level functions for making requests.

getCurlHandle *Create libcurl handles*

Description

These functions create and duplicate curl handles for use in calls to the HTTP facilities provided by that low-level language and this R-level interface. A curl handle is an opaque data type that contains a reference to the internal C-level data structure of libcurl for performing HTTP requests.

The getCurlMutliHandle returns an object that can be used for concurrent, multiple requests. It is quite different from the regular curl handle and again, should be treated as an opaque data type.

Usage

```
getCurlHandle(..., .opts = NULL, .encoding = integer(),
              .defaults = getOption("RCurlOptions"))
dupCurlHandle(curl, ..., .opts = NULL, .encoding = integer())
getCurlMultiHandle(..., .handles = list(...))
```

Arguments

curl	the existing curl handle that is to be duplicated
...	a named list of curl options to set after the handle has been created. For getCurlMultiHandle, these values are invidivual curl handle objects, created via getCurlHandle or dupCurlHandle.
.opts	a named list or CURLOptions object identifying the curl options for the handle. These and the ... arguments are used after the handle has been created.
.encoding	an integer or a string that explicitly identifies the encoding of the content that is returned by the HTTP server in its response to our query. The possible strings are 'UTF-8' or 'ISO-8859-1' and the integers should be specified symbolically as CE_UTF8 and CE_LATIN1. Note that, by default, the package attempts to process the header of the HTTP response to determine the encoding. This argument is used when such information is erroneous and the caller knows the correct encoding.
.defaults	a collection of default values taken from R's global/session options. This is a parameter so that one can override it if necessary.
.handles	a list of curl handle objects that are used as the individual request handles within the multi-asynchronous requests

Details

These functions create C-level data structures.

Value

An object of class CURLHandle which is simply a pointer to the memory for the C structure.

Author(s)

Duncan Temple Lang

References

Curl homepage https://curl.se/

See Also

getURL curlPerform

Examples

```
options(RCurlOptions = list(verbose = TRUE,
                            followlocation = TRUE,
                            autoreferer = TRUE,
                            nosignal = TRUE))
if(url.exists("https://r-project.org")) {
   x = getURL("https://r-project.org")
      # here we override one of these.
   x = getURL("https://r-project.org", verbose = FALSE)
}
```

---

getCurlInfo *Access information about a CURL request*

Description

This function provides access to data about a previously executed CURL request that is accessible via a CURLHandle object. This means, of course, that one must have access to the CURLHandle object. The information one can get includes items such as the name of the file (potentially containing redirects), download time,

See getCurlInfoConstants for the names of the possible fields.

Usage

```
getCurlInfo(curl, which = getCurlInfoConstants())
getCurlInfoConstants()
```

Arguments

curl	the CURLHandle object used to perform the request. This is a reference to an opaque internal C-level data structure that is provided and used by libcurl to make a request.
which	identifiers for the elements of interest. These can be specified by integer value or by name. The names are matched against those in the getCurlInfoConstants

.

Details

This is an interface to the `get_curl_info` routine in the libcurl package.

Value

A named list whose elements correspond to the requested fields. The names are expanded to match the names of these fields and the values are either strings or integer values.

Author(s)

Duncan Temple Lang

References

Curl homepage https://curl.se/

See Also

`curlPerform` `getURL` `getCurlHandle`

Examples

```
tryCatch(withAutoprint({
   curl = getCurlHandle()
   txt = getURL("https://r-project.org", curl = curl)
   getCurlInfo(curl)
   rm(curl)  # release the curl!
}), GenericCurlError = identity)
```

---

getFormParams *Extract parameters from a form query string*

Description

This function facilitates getting the parameter names and values from a URL that is an parameterized HTML query.

This is motivated by a function from Chris Davis and Delft University.

Usage

```
getFormParams(query, isURL = grepl("^(http|\\?)", query))
```

Arguments

`query`	the query string or full URL containing the query
`isURL`	a logical value. If `TRUE`, `query` is the full URL and we need to extract the substring representing the parameters. If `isURL` is `FALSE`, then `query` is assumed to be just the string containing the parameters.

Value

A named character vector giving the parameter values The names are the parameter names.

Author(s)

Duncan Temple Lang

Examples

```
getFormParams("https://example.invalid/foo/bob.R?xyz=1&abc=verylong")

getFormParams("xyz=1&abc=verylong")
getFormParams("xyz=1&abc=&on=true")
getFormParams("xyz=1&abc=")
```

---

getURIAsynchronous *Download multiple URIs concurrently, with inter-leaved downloads*

Description

This function allows the caller to specify multiple URIs to download at the same time. All the requests are submitted and then the replies are processed as data becomes available on each connection. In this way, the responses are processed in an inter-leaved fashion, with a chunk from one response from one request being processed and then followed by a chunk from a different request.

Downloading documents asynchronously involves some trade-offs. The switching between different streams, detecting when input is available on any of them involves a little more processing and so increases the consumption of CPU cycles. On the other hand, there is a potentially large saving of time when one considers total time to download. See https://aitap.grebedoc.dev/RCurl/concurrent.html for more details. This is a common trade-off that arises in concurrent/parallel/asynchronous computing.

getURI calls this function if more than one URI is specified and async is TRUE, the default in this case. One can also download the (contents of the) multiple URIs serially, i.e. one after the other using getURI with a value of FALSE for async.

Usage

```
getURIAsynchronous(url, ..., .opts = list(), write = NULL,
                   curl = getCurlHandle(),
                   multiHandle = getCurlMultiHandle(), perform = Inf,
                  .encoding = integer(), binary = rep(NA, length(url)))
```

Arguments

url	a character vector identifying the URIs to download.
...	named arguments to be passed to curlSetOpt when creating each of the different curlHandle objects.

.opts	a named list or CURLOptions object identifying the curl options for the handle. This is merged with the values of ... to create the actual options for the curl handle in the request.
write	an object giving the functions or routines that are to be called when input is waiting on the different HTTP response streams. By default, a separate callback function is associated with each input stream. This is necessary for the results to be meaningful as if we use a single reader, it will be called for all streams in a haphazard order and the content interleaved. One can do interesting things however using a single object.
curl	the prototypical curlHandle that is duplicated and used in in
multiHandle	this is a curl handle for performing asynchronous requests.
perform	a number which specifies the maximum number of calls to curlMultiPerform that are to be made in this function call. This is typically either 0 for no calls or Inf meaning process the requests until completion. One may find alternative values useful, such as 1 to ensure that the requests are dispatched.
.encoding	an integer or a string that explicitly identifies the encoding of the content that is returned by the HTTP server in its response to our query. The possible strings are 'UTF-8' or 'ISO-8859-1' and the integers should be specified symbolically as CE_UTF8 and CE_LATIN1. Note that, by default, the package attempts to process the header of the HTTP response to determine the encoding. This argument is used when such information is erroneous and the caller knows the correct encoding.
binary	a logical vector identifying whether each URI has binary content or simple text.

Details

This uses curlMultiPerform and the multi/asynchronous interface for libcurl.

Value

The return value depends on the run-time characteristics of the call. If the call merely specifies the URIs to be downloaded, the result is a named character vector. The names identify the URIs and the elements of the vector are the contents of the corresponding URI.

If the requests are not performed or completed (i.e. perform is zero or too small a value to process all the chunks) a list with 2 elements is returned. These elements are:

multiHandle	the curl multi-handle, of class MultiCURLHandle-class. This can be used in further calls to curlMultiPerform
write	the write argument (after it was potentially expanded to a list). This can then be used to fetch the results of the requests when the requests are completed in the future.

Author(s)

Duncan Temple Lang <duncan@r-project.org>

References

Curl homepage https://curl.se/

See Also

getURL getCurlMultiHandle curlMultiPerform

Examples

```
uris = c("https://r-project.org",
         "https://aitap.grebedoc.dev/RCurl/philosophy.xml")
tryCatch(withAutoprint({
  txt = getURIAsynchronous(uris, maxredirs = 20)
  names(txt)
  nchar(txt)
}), GenericCurlError = identity)
```

---

getURL *Download a URI*

Description

These functions download one or more URIs (a.k.a. URLs). It uses libcurl under the hood to perform the request and retrieve the response. There are a myriad of options that can be specified using the ... mechanism to control the creation and submission of the request and the processing of the response.

getURLContent has been added as a high-level function like getURL and getBinaryURL but which determines the type of the content being downloaded by looking at the resulting HTTP header's Content-Type field. It uses this to determine whether the bytes are binary or "text".

The request supports any of the facilities within the version of libcurl that was installed. One can examine these via curlVersion.

getURLContent doesn't perform asynchronous or multiple concurrent requests at present.

Usage

```
getURL(url, ..., .opts = list(),
        write = basicTextGatherer(.mapUnicode = .mapUnicode),
         curl = getCurlHandle(), async = length(url) > 1,
           .encoding = integer(), .mapUnicode = TRUE)
getURI(url, ..., .opts = list(),
        write = basicTextGatherer(.mapUnicode = .mapUnicode),
         curl = getCurlHandle(), async = length(url) > 1,
          .encoding = integer(), .mapUnicode = TRUE)
getURLContent(url, ..., curl = getCurlHandle(.opts = .opts), .encoding = NA,
              binary = NA, .opts = list(...),
              header = dynCurlReader(curl, binary = binary,
```

```
                                    baseURL = url, isHTTP = isHTTP,
                                     encoding = .encoding),
                isHTTP = length(grep('^[[:space:]]*http', url)) > 0)
```

Arguments

url	a string giving the URI
...	named values that are interpreted as CURL options governing the HTTP request.
.opts	a named list or CURLOptions object identifying the curl options for the handle. This is merged with the values of ...to create the actual options for the curl handle in the request.
write	if explicitly supplied, this is a function that is called with a single argument each time the the HTTP response handler has gathered sufficient text. The argument to the function is a single string. The default argument provides both a function for cumulating this text and is then used to retrieve it as the return value for this function.
curl	the previously initialized CURL context/handle which can be used for multiple requests.
async	a logical value that determines whether the download request should be done via asynchronous, concurrent downloading or a serial download. This really only arises when we are trying to download multiple URIs in a single call. There are trade-offs between concurrent and serial downloads, essentially trading CPU cycles for shorter elapsed times. Concurrent downloads reduce the overall time waiting for getURI/getURL to return.
.encoding	an integer or a string that explicitly identifies the encoding of the content that is returned by the HTTP server in its response to our query. The possible strings are 'UTF-8' or 'ISO-8859-1' and the integers should be specified symbolically as CE_UTF8 and CE_LATIN1. Note that, by default, the package attempts to process the header of the HTTP response to determine the encoding. This argument is used when such information is erroneous and the caller knows the correct encoding. The default value leaves the decision to this default mechanism. This does however currently involve processing each line/chunk of the header (with a call to an R function). As a result, if one knows the encoding for the resulting response, specifying this avoids this slight overhead which is probably quite small relative to network latency and speed.
.mapUnicode	a logical value that controls whether the resulting text is processed to map components of the form \uxxxx to their appropriate Unicode representation.
binary	a logical value indicating whether the caller knows whether the resulting content is binary (TRUE) or not (FALSE) or unknown (NA).
header	this is made available as a parameter of the function to allow callers to construct different readers for processing the header and body of the (HTTP) response. Callers specifying this will typically only adjust the call to dynCurlReader, e.g. to specify a function for its value parameter to control how the body is post-processed. The caller can specify a value of TRUE or FALSE for this parameter. TRUE means that the header will be returned along with the body; FALSE corresponds to the

default and only the body will be returned. When returning the header, it is first parsed via parseHTTPHeader, unless the value of header is of class AsIs. So to get the raw header, pass the argument as header = I(TRUE).

isHTTP — a logical value that indicates whether the request an HTTP request. This is used when determining how to process the response.

Value

If no value is supplied for write, the result is the text that is the HTTP response. (HTTP header information is included if the header option for CURL is set to TRUE and no handler for headerfunction is supplied in the CURL options.)

Alternatively, if a value is supplied for the write parameter, this is returned. This allows the caller to create a handler within the call and get it back. This avoids having to explicitly create and assign it and then call getURL and then access the result. Instead, the 3 steps can be inlined in a single call.

Author(s)

Duncan Temple Lang

References

Curl homepage https://curl.se/

See Also

getBinaryURL curlPerform curlOptions

Examples

```
 # Regular HTTP
if(requireNamespace("XML", quietly = TRUE)) tryCatch(withAutoprint({
   txt = getURL("https://r-project.org/", maxredirs = 20)
   ## Then we could parse the result.
   XML::htmlTreeParse(txt, asText = TRUE)
}), GenericCurlError = identity)


      # HTTPS. First check to see that we have support compiled into
      # libcurl for ssl.
if(interactive() && ("ssl" %in% names(curlVersion()$features))
       && url.exists("https://sourceforge.net/")) {
   txt = tryCatch(getURL("https://sourceforge.net/"),
                  error = function(e) {
                              getURL("https://sourceforge.net/",
                                        ssl.verifypeer = FALSE)
                          })

}


   # Create a CURL handle that we will reuse.
```

```
  if(interactive()) {
     curl = getCurlHandle(maxredirs = 20)
     pages = list()
     for(u in c("https://r-project.org",
                "https://aitap.grebedoc.dev/RCurl/concurrent.html")) {
         pages[[u]] = tryCatch(getURL(u, curl = curl),
                               GenericCurlError = identity)
     }
  }


    # Set additional fields in the header of the HTTP request.
    # verbose option allows us to see that they were included.
  tryCatch(withAutoprint(
    getURL("https://r-project.org", httpheader = c(Accept = "text/html",
                                                   MyField = "Duncan"),
           verbose = TRUE)
  ), GenericCurlError = identity)



    # Arrange to read the header of the response from the HTTP server as
    # a separate "stream". Then we can break it into name-value
    # pairs. (The first line is the HTTP/1.1 200 Ok or 301 Moved Permanently
    # status line)
  tryCatch(withAutoprint({
     h = basicTextGatherer()
     txt = getURL("https://www.r-project.org",
                  header= TRUE, headerfunction = h$update,
                  httpheader = c(Accept="text/html", Test=1), verbose = TRUE)
     print(paste(h$value(NULL)[-1], collapse=""))
     con <- textConnection(paste(h$value(NULL)[-1], collapse=""))
     read.dcf(con)
     close(con)
  }), GenericCurlError = identity)



## Not run:
  # Test the passwords.
  x = getURL("https://example.invalid/RCurl/testPassword/index.html",  userpwd = "bob:duncantl")

  # Catch an error because no authorization
  # We catch the generic HTTPError, but we could catch the more specific "Unauthorized" error
  # type.
  x = tryCatch(getURLContent("https://example.invalid/RCurl/testPassword/index.html"),
               HTTPError = function(e) {
                              cat("HTTP error: ", e$message, "\n")
                           })

## End(Not run)

## Not run:
```

```
  #  Needs specific information from the cookie file on a per user basis
  #  with a registration to the NY times.
  x = getURL("https://www.nytimes.com",
                header = TRUE, verbose = TRUE,
                cookiefile = "/home/duncan/Rcookies",
                netrc = TRUE,
                maxredirs = as.integer(20),
                netrc.file = "/home2/duncan/.netrc1",
                followlocation = TRUE)

## End(Not run)

   if(interactive()) tryCatch(withAutoprint({
       d = debugGatherer()
       x = getURL("https://r-project.org", debugfunction = d$update, verbose = TRUE)
       d$value()
   }), GenericCurlError = identity)

## Not run:
   ###########################################
   #  Using an option set in R

   opts = curlOptions(header = TRUE, userpwd = "bob:duncantl", netrc = TRUE)
  getURL("https://example.invalid/RCurl/testPassword/index.html", verbose = TRUE, .opts = opts)

      # Using options in the CURL handle.
   h = getCurlHandle(header = TRUE, userpwd = "bob:duncantl", netrc = TRUE)
  getURL("https://example.invalid/RCurl/testPassword/index.html", verbose = TRUE, curl = h)

## End(Not run)


   # Use a C routine as the reader. Currently gives a warning.
  if(interactive()) tryCatch(withAutoprint({
     routine = getNativeSymbolInfo("R_internalWriteTest", PACKAGE = "RCurl")$address
     getURL("https://r-project.org", writefunction = routine)
  }), GenericCurlError = identity)



  # Example
  if(interactive()) {
     uris = c("https://r-project.org",
              "https://aitap.grebedoc.dev/RCurl/philosophy.xml")
     txt = tryCatch(getURI(uris), GenericCurlError = conditionMessage)
     names(txt)
     nchar(txt)

     txt = tryCatch(getURI(uris, async = FALSE),
                    GenericCurlError = conditionMessage)
     names(txt)
     nchar(txt)
```

```
  routine = getNativeSymbolInfo("R_internalWriteTest", PACKAGE = "RCurl")$address
  txt = tryCatch(getURI(uris, write = routine, async = FALSE),
                 GenericCurlError = conditionMessage)
  names(txt)
  nchar(txt)


     # getURLContent() for text and binary
  x = tryCatch(getURLContent("https://r-project.org"),
               GenericCurlError = identity)
  class(x)

  x = tryCatch(getURLContent("https://aitap.codeberg.page/RCurl/data.gz"),
               GenericCurlError = identity)
  class(x)
  attr(x, "Content-Type")

  x = tryCatch(getURLContent("https://www.r-project.org/Rlogo.png"),
               GenericCurlError = identity)
  class(x)
  attr(x, "Content-Type")


  curl = getCurlHandle()
  dd = tryCatch(getURLContent("https://aitap.grebedoc.dev/RCurl/RJSONIO.pdf",
                              curl = curl,
                              header = dynCurlReader(curl, binary = TRUE,
                                                     value = function(x) {
                                                              print(attributes(x))
                                                              x})),
                GenericCurlError = identity)
}



# FTP
# Download the files within a directory.
if(interactive() && url.exists('ftp://ftp.wcc.nrcs.usda.gov')) {

  url = 'ftp://ftp.wcc.nrcs.usda.gov/data/snow/snow_course/table/history/idaho/'
  filenames = getURL(url, ftp.use.epsv = FALSE, dirlistonly = TRUE)

    # Deal with newlines as \n or \r\n. (BDR)
    # Or alternatively, instruct libcurl to change \n's to \r\n's for us with crlf = TRUE
    # filenames = getURL(url, ftp.use.epsv = FALSE, ftplistonly = TRUE, crlf = TRUE)
  filenames = paste(url, strsplit(filenames, "\r*\n")[[1]], sep = "")
  con = getCurlHandle( ftp.use.epsv = FALSE)

    # there is a slight possibility that some of the files that are
    # returned in the directory listing and in filenames will disappear
    # when we go back to get them. So we use a try() in the call getURL.
  contents = sapply(filenames[1:5], function(x) try(getURL(x, curl = con)))
```

```
   names(contents) = filenames[1:length(contents)]
}
```

---

guessMIMEType *Infer the MIME type from a file name*

Description

This function returns the MIME type, i.e. part of the value used in the Content-Type for an HTTP request/response or in email to identify the nature of the content. This is a string such as "text/plain" or "text/xml" or "image/png".

The function consults an R object constructed by reading a Web site of known MIME types (not necessarily all) and matching the extension of the file name to the names of that table.

Usage

```
guessMIMEType(name, default = NA)
```

Arguments

name	character vector of file names
default	the value to use if no MIME type is found in the table for the given file name/extension.

Value

A character vector giving the MIME types for each element of name.

Author(s)

Duncan Temple Lang

References

The table of MIME types and extensions was programmatically extracted from 'http://www.webmaster-toolkit.com/mim with tbls = readHTMLTable("http://www.webmaster-toolkit.com/mime-types.shtml") tmp = tbls[[1]][-1,] mimeTypeExtensions = structure(as.character(tmp[[2]]), names = gsub("^\.", "", tmp[[1]])) save(mimeTypeExtensions, file = "data/mimeTypeExtensions.rda") The IANA list is not as convenient to programmatically compile.

See Also

Uploading file.

Examples

```
guessMIMEType(c("foo.txt", "foo.png", "foo.jpeg", "foo.Z", "foo.R"))

guessMIMEType("foo.bob")
guessMIMEType("foo.bob", "application/x-binary")
```

`httpPUT` *Simple high-level functions for HTTP PUT and DELETE*

Description

These two functions are simple, high-level functions that implement the HTTP request methods PUT and DELETE. These can also be done by specifying the method type using the curl option `customrequest`. These functions merely provide a convenience wrapper for `getURLContent` with the HTTP method specified.

Usage

```
httpPUT(url, content,  ..., curl = getCurlHandle())
httpPOST(url, ..., curl = getCurlHandle())
httpDELETE(url, ..., curl = getCurlHandle())
httpGET(url, ..., curl = getCurlHandle())
httpHEAD(url, ..., curl = getCurlHandle())
httpOPTIONS(url, ..., curl = getCurlHandle())
```

Arguments

`url`	the URL of the server to which the HTTP request is to be made
`content`	the value that is to be used as the content of the `PUT` request. This can be a character or a `raw` object.
`...`	additional arguments passed to `getURLContent`
`curl`	the curl handle to be used to make the request

Value

The content returned by the server as a result of the request.

Author(s)

Duncan Temple Lang

See Also

`getURLContent`

Examples

```
## Not run:
   # create a database in a CouchDB server
 httpPUT("http://127.0.0.1:5984/temp_db")

   # Insert an entry into an ElasticSearch dabtabase.
 httpPUT("http://localhost:9200/a/b/axyz", '{"abc" : 123}')

   # Then delete the database
 httpDELETE("http://127.0.0.1:5984/temp_db")

## End(Not run)
```

HTTP_VERSION_1_0 *Symbolic constants for specifying HTTP and SSL versions in libcurl*

Description

These are values that can be used to set the `http.version` and `sslversion` options of `curlPerform`.

Usage

```
HTTP_VERSION_1_0
```

References

https://curl.se/libcurl/c/curl_easy_setopt.html

---

merge.list *Method for merging two lists by name*

Description

This is a method that merges the contents of one list with another by adding the named elements in the second that are not in the first. In other words, the first list is the target template, and the second one adds any extra elements that it has.

Usage

```
merge.list(x, y, ...)
```

Arguments

`x`	the list to which elements will be added
`y`	the list which will supply additional elements to `x` that are not already there by name.
`...`	not used.

Value

A named list whose name set is the union of the elements in names of x and y and whose values are those taken from y and then with those in x, overwriting if necessary.

Author(s)

Duncan Temple Lang

References

Curl homepage https://curl.se/

See Also

merge

Examples

```
## Not run:
# Not exported.

 merge.list(list(a=1, b = "xyz", c = function(x, y) {x+y}),
            list(a = 2, z = "a string"))

  # No values in y
 merge.list(list(a=1, b = "xyz", c = function(x, y) {x+y}), list())

  # No values in x
 merge.list(list(), list(a=1, b = "xyz", c = function(x, y) {x+y}))

## End(Not run)
```

---

mimeTypeExtensions *Mapping from extension to MIME type*

Description

This is a programmatically generated character vector whose names identify the MIME type typically associated with the extension which are the values. This is used in guessMIMEType. We can match an extension and then find the corresponding MIME type. There are duplicates.

Usage

```
data(mimeTypeExtensions)
```

Format

The format is a named character vector where the names are the MIME types and the values are the file extensions.

Source

The table of MIME types and extensions was programmatically extracted from 'http://www.webmaster-toolkit.com/mim with `tbls = readHTMLTable("http://www.webmaster-toolkit.com/mime-types.shtml") tmp = tbls[[1]][-1,] mimeTypeExtensions = structure(as.character(tmp[[2]]), names = gsub("^\.", "", tmp[[1]])) save(mimeTypeExtensions, file = "data/mimeTypeExtensions.rda")` The IANA list is not as convenient to programmatically compile.

Examples

```
data(mimeTypeExtensions)
```

`MultiCURLHandle-class` *Class "MultiCURLHandle" for asynchronous, concurrent HTTP requests*

Description

This is a class that represents a handle to an internal C-level data structure provided by libcurl to perform multiple HTTP requests in a single operation and process the responses in an inter-leaved fashion, i.e. a chunk from one, followed by a chunk from another.

Objects of this class contain not only a reference to the internal C-level data structure, but also have a list of the `CURLHandle-class` objects that represent the individual HTTP requests that make up the collection of concurrent requests. These are maintained for garbage collection reasons.

Essentially, the data in objects of this class are for internal use; this is an opaque class in R.

Objects from the Class

The constructor function `getCurlMultiHandle` is the only way to create meaningful instances of this class.

Slots

`ref`: Object of class `"externalptr"`. This is a reference to the instance of the libcurl data structure `CURLM` pointer.

`subhandles`: Object of class `"list"`. This is a list of `CURLHandle-class` instances that have been `push()`ed onto the multi-handle stack.

Methods

pop `signature(obj = "MultiCURLHandle", val = "CURLHandle")`: ...

pop `signature(obj = "MultiCURLHandle", val = "character")`: ...

push `signature(obj = "MultiCURLHandle", val = "CURLHandle")`: ...

Author(s)

Duncan Temple Lang

References

Curl homepage https://curl.se/

See Also

getCurlMultiHandle curlMultiPerform multiTextGatherer

---

postForm *Submit an HTML form*

Description

These functions provide facilities for submitting an HTML form using either the simple GET mechanism (appending the name-value pairs of parameters in the URL) or the POST method which puts the name-value pairs as separate sections in the body of the HTTP request. The choice of action is defined by the form, not the caller.

Usage

```
postForm(uri, ..., .params = list(), .opts = curlOptions(url = uri),
         curl = getCurlHandle(), style = 'HTTPPOST',
          .encoding = integer(), binary = NA, .checkParams = TRUE,
          .contentEncodeFun = curlEscape)
.postForm(curl, .opts, .params, style = 'HTTPPOST')
getForm(uri, ..., .params = character(), .opts = list(), curl = getCurlHandle(),
         .encoding = integer(), binary = NA, .checkParams = TRUE)
```

Arguments

uri	the full URI to which the form is to be posted. This includes the host and the specific file or script which will process the form.
...	the name-value pairs of parameters. Note that these are not the CURL options.
.params	instead of specifying the name-value parameters in "free" form via the ... argument, one can specify them as named list or character vector.
.opts	an object representing the CURL options for this call.
curl	the CURLHandle object created earlier if one is reusing these objects. Otherwise, a new one is generated and discarded.
style	this is typically a string and controls how the form data is posted, specifically the value for the Content-Type header and the particular representation. Use 'http-post' to use a multipart/form-data transmission and use 'post' for application/x-www-form-urlenc content. This string is compared to the names of (the internal) PostStyles vector using partial matching. In the future, we will use enum values within R. The default is multipart/form-data for reasons of backward compatability.
.encoding	the encoding of the result, if it is known a priori. This can be an integer between 0 and 4 or more appropriately a string identifying the encoding as one of "utf-8", or "ISO-859-1".

`binary` a logical value indicating whether the caller knows whether the resulting content is binary (`TRUE`) or not (`FALSE`) or unknown (`NA`).

`.checkParams` a logical value that indicates whether we should perform a check/test to identify if any of the arguments passed to the form correspond to Curl options. This is useful to identify potential errors in specifying the Curl options in the wrong place (in the way we would for `curlPerform`). This check can lead to spurious warning messages if the form has parameters with names that do conflict with Curl options. By specifying `FALSE` for this parameter, you can avoid this test and hence any warnings. But make certain you know what you are doing.

`.contentEncodeFun` a function which encodes strings in a suitable manner. For x-www-form-encoded submissions, this should most likely should be `curlPercentEncode` which maps spaces to +, = to %3D, etc. We are leaving the default as `curlEscape` for now until we test whether applications continue to work with the correct encoding.

Details

Creating a new `CURLHandle` allows the C-level code to more efficiently map the R-level values to their C equivalents needed to make the call. However, reusing the handle across calls can be more efficient in that the connection to a server can be maintained and thus, the sometimes expensive task of establishing it is avoided in subsequent calls.

Value

By default, the text from the HTTP response is returned.

See Also

`getURL` `curlOptions` `curlSetOpt`

Examples

```
if(url.exists("http://www.google.com")) withAutoprint({
   # Two ways to submit a query to google. Searching for RCurl
  getURL("http://www.google.com/search?hl=en&lr=&ie=ISO-8859-1&q=RCurl&btnG=Search")

   # Here we let getForm do the hard work of combining the names and values.
  getForm("http://www.google.com/search", hl="en", lr="",
          ie="ISO-8859-1", q="RCurl", btnG="Search")

   # And here if we already have the parameters as a list/vector.
  getForm("http://www.google.com/search", .params = c(hl="en", lr="",
          ie="ISO-8859-1",  q="RCurl", btnG="Search"))
})


   # Now looking at POST method for forms.
url <- "http://wwwx.cs.unc.edu/~jbs/aw-wwwp/docs/resources/perl/perl-cgi/programs/cgi_stdin.cgi"
if(url.exists(url))
 postForm(url,
```

```
            name = "Bob", "checkedbox" = "spinich",
            submitButton = "Now!",
            textarea = "Some text to send",
            selectitem = "The item",
            radiobutton = "a", style = "POST")


  # Genetic database via the Web.
if(url.exists('http://www.wormbase.org/db/searches/advanced/dumper')) withAutoprint({
 x = postForm('http://www.wormbase.org/db/searches/advanced/dumper',
         species="briggsae",
         list="",
         flank3="0",
         flank5="0",
         feature="Gene Models",
         dump = "Plain TEXT",
         orientation = "Relative to feature",
         relative = "Chromsome",
         DNA ="flanking sequences only",
      .cgifields = paste(c("feature", "orientation", "DNA", "dump","relative"), collapse=", "))


 # Note that we don't have to paste multiple values together ourselves,
 # e.g. the .cgifields can be specified as a character vector rather
 # than a string.
 x = postForm('http://www.wormbase.org/db/searches/advanced/dumper',
         species="briggsae",
         list="",
         flank3="0",
         flank5="0",
         feature="Gene Models",
         dump = "Plain TEXT",
         orientation = "Relative to feature",
         relative = "Chromsome",
         DNA ="flanking sequences only",
         .cgifields =c("feature", "orientation", "DNA", "dump", "relative"))
})
```

RCurlInternal *Internal functions*

Description

Not for human consumption

`reset` *Generic function for resetting an object*

Description

This generic and the associated method for a CURLHandle allows one to reset the state of the Curl object to its default state. This is convenient if we want to reuse the same connection, but want to ensure that it is in a particular state.

Unfortunately, we cannot query the state of different fields in an existing Curl handle and so we need to be able to reset the state and then update it with any particular settings we would have liked to keep.

Usage

```
reset(x, ...)
```

Arguments

`x`	the object to be reset. For our method, this is an object of class `CURLHandle`.
`...`	additional arguments for methods

Details

This calls the C routine `curl_easy_reset` in libcurl.

Value

Methods typically return the updated version of the object passed to it. This allows the caller to assign the new result to the same variable rather than relying on mutating the content of the object in place. In other words, the object should not be treated as a reference but a new object with the updated contents should be returned.

Author(s)

Duncan Temple Lang

References

Curl homepage https://curl.se/

See Also

`getCurlHandle` `dupCurlHandle`

Examples

```
h = getCurlHandle()
curlSetOpt(customrequest = "DELETE")
reset(h)
```

scp *Retrieve contents of a file from a remote host via SCP (Secure Copy)*

Description

This function allows us to retrieve the contents of a file from a remote host via SCP. This is done entirely within R, rather than a command line application and the contents of the file are never written to disc. The function allows the

Usage

```
scp(host, path, keypasswd = NA, user = getUserName(), rsa = TRUE,
     key = sprintf(c("~/.ssh/id_%s.pub", "~/.ssh/id_%s"),
                   if (rsa) "rsa" else "dsa"),
     binary = NA, size = 5000, curl = getCurlHandle(), ...)
```

Arguments

host	the name of the remote host or its IP address
path	the path of the file of interest on the remote host's file systems
keypasswd	a password for accessing the local SSH key. This is the passphrase for the key.
user	the name of the user on the remote machine
rsa	a logical value indicating whether to use the RSA or DSA key
key	the path giving the location of the SSH key.
binary	a logical value giving
size	an estimate of the size of the buffer needed to store the contents of the file. This is used to initialize the buffer and potentially avoid resizing it as needed.
curl	a curl handle (getCurlHandle) that is to be reused for this request and which potentially contains numerous options settings or an existing connection to the host.
...	additional parameters handed to curlPerform.

Details

This uses libcurl's facilities for scp. Use `"scp" %in% curlVersion()$protocols` to see if SCP is supported.

Value

Either a raw or character vector giving the contents of the file.

Author(s)

Duncan Temple Lang

References

libcurl https://curl.se/

See Also

curlPerform getCurlOptionsConstants

Examples

```
## Not run:
   x = scp("eeyore.ucdavis.edu", "/home/duncan/OmegaWeb/index.html",
           "My.SCP.Passphrase", binary = FALSE)
   x = scp("eeyore.ucdavis.edu", "/home/duncan/OmegaWeb/RCurl/xmlParse.bz2",
           "My.SCP.Passphrase")
   o = memDecompress(x, asChar = TRUE)

## End(Not run)
```

---

url.exists *Check if URL exists*

Description

This functions is analogous to file.exists and determines whether a request for a specific URL responds without error. We make the request but ask the server not to return the body. We just process the header.

Usage

```
url.exists(url, maxredirs = 20, ...,
           .opts = list(maxredirs = maxredirs, ...),
           curl = getCurlHandle(.opts = .opts), .header = FALSE)
```

Arguments

url	a vector of one or more URLs whose existence we are to test
maxredirs	The maximal number of redirects. Set to 20 in order to fail redirect loops.
...	name = value pairs of Curl options.
.opts	a list of name = value pairs of Curl options.
curl	a Curl handle that the caller can specify if she wants to reuse an existing handle, e.g. with different options already specified or that has previously established a connection to the Web server
.header	a logical value that if TRUE causes the header information to be returned.

Details

This makes an HTTP request but with the nobody option set to `FALSE` so that we don't actually retrieve the contents of the URL.

Value

If `.header` is `FALSE`, this returns `TRUE` or `FALSE` for each URL indicating whether the request was successful (had a status with a value in the 200 range).

If `.header` is `TRUE`, the header is returned for the request for each URL.

Author(s)

Duncan Temple Lang

References

HTTP specification

See Also

`curlPerform`

Examples

```
url.exists("https://r-project.org")
try(url.exists("http://example.invalid"))
```

Index