Package ‘base64enc’

February 2, 2026

Version 0.1-6

Title Tools for 'base64' Encoding

Author Simon Urbanek [aut, cre, cph] (https://urbanek.nz, ORCID: <https://orcid.org/0000-0003-2297-1732>)

Maintainer Simon Urbanek <Simon.Urbanek@r-project.org>

Depends R (>= 2.9.0)

Enhances png

Description
Tools for handling 'base64' encoding. It is more flexible than the orphaned 'base64' package.

License GPL-2 | GPL-3

URL https://www.rforge.net/base64enc

BugReports https://github.com/s-u/base64enc/issues

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-02-02 06:31:10 UTC

Contents

base64 2
checkUTF8 3
dataURI 4

Index **5**

base64 *Encode/decode data into/from base64 encoding*

Description

`base64encode` encodes a data into base64 encoding. The source can be a file, binary connection or a raw vector.

`base64decode` decodes a base64-encoded string into binary data. The source can be a string or a connection, the output is either a raw vector (`output=NULL`) or a binary connection.

Usage

```
base64encode(what, linewidth, newline)
base64decode(what, output = NULL, file, strict = FALSE)
```

Arguments

`what`	data to be encoded/decoded. For `base64encode` it can be a raw vector, text connection or file name. For `base64decode` it can be a string, raw vector or a binary connection.
`linewidth`	if set, the output is split into lines with at most `linewidth` characters per line. Zero or `NA` denotes no limit and values 1 .. 3 are silently treated as 4 since that is the shortest valid line.
`newline`	only applicable if `linewidth` is set; if set (string), the result will be a single string with all lines joined using the `newline` string
`output`	if `NULL` then the output will be a raw vector with the decoded data, otherwise it must be either a filename (string) or a binary connection.
`file`	file name (string) for data to use as input instead of `what`. It is essentially just a shorthand for `base64decode(file(name))`. Only one of `what` and `file` can be specified.
`strict`	logical scalar. If `TRUE` then the decoder validates the input contents making sure it strictly adheres to the standard, does not include any other characters (including white spaces, newlines etc.), is correctly padded and does not have any trailing content. Any validation failure will result in an error. Otherwise the decoding skips over invalid characters, permits lack of padding and ignores trailing content.

Value

`base64encode`: A character vector. If `linewith > 0` and `newline` is not set then it will consist of as many elements as there are lines. Otherwise it is a single string.

`base64decode`: If `output = NULL` then a raw vector with the decoded content, otherwise the number of bytes written into the connection.

Author(s)

Simon Urbanek

Examples

```
base64encode(1:100)
base64encode(1:100, 70)
base64encode(1:100, 70, "\n")
x <- charToRaw("the decoded content, otherwise the number of bytes")
y <- base64decode(base64encode(x))
stopifnot(identical(x, y))
```

`checkUTF8` *Check the validity of a byte stream ot be interpreted as UTF8.*

Description

`checkUTF8` check whether a given raw vector can be used as a valid string encoded in UTF8.

Usage

```
checkUTF8(what, quiet = FALSE,charlen = FALSE, min.char = 1L)
```

Arguments

`what`	raw vector with the payload
`quiet`	logical, if `TRUE` then the function will not fail but report success/failure via its result, otherwise failures are considered errors.
`charlen`	logical, if `TRUE` then the function returns the length of the longest byte sequence representing a character in the file.
`min.char`	integer, any bytes below this value are considered control chacters and reported as errors. The default value of 1L guards against strings including NULs.

Value

If `charlen=FALSE`: `TRUE` on success, `FALSE` if the payload is invalid and `quite=TRUE`.

If `charlen=TRUE`: positive integer corresponding to the longest encoded sequence on success, negative integer on failure.

Author(s)

Simon Urbanek

`dataURI` *Create a data URI string*

Description

`dataURI` creates URI with the `data:` scheme by encoding the payload either using base64 ot URI encoding.

Usage

```
dataURI(data, mime = "", encoding = "base64", file)
```

Arguments

`data`	raw vector, connection or character vector to use as payload. Character vectors of more than one element are collapsed using `"\n"` before encoding.
`mime`	MIME-type of the data (per standard "" is interpreted as "text/plain;charset=US-ASCII" without including it in the URI)
`encoding`	data encoding to use. Must be either `"base64"` or `NULL`
`file`	filename (string) to open as payload. `file` and `data` are mutually exclusive

Value

string of the form `data:[mime][;base64],<encoded-payload>`

Author(s)

Simon Urbanek

References

RFC 2397 The "data" URL scheme

Examples

```
dataURI(as.raw(1:10)) # default is base64
dataURI(as.raw(1:10), encoding=NULL) # URI
if (require("png", quietly=TRUE)) {
  # let's say you have an image - e.g. from dev.capture(TRUE)
  img <- matrix(1:16/16, 4)
  dataURI(writePNG(img), "image/png")
  # or straight from a file
  dataURI(file=system.file("img", "Rlogo.png", package="png"), mime="image/png")
}
```

Index

∗ **manip**
  base64, 2
  checkUTF8, 3
  dataURI, 4

base64, 2
base64decode *(*base64*)*, 2
base64encode *(*base64*)*, 2

checkUTF8, 3

dataURI, 4