Package 'jpeg'

November 29, 2022

Version 0.1-10

Title Read and write JPEG images

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

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

Depends R (>= 2.9.0)

Description This package provides an easy and simple way to read, write and display bitmap images stored in the JPEG format. It can read and write both files and in-memory raw vectors.

License GPL-2 | GPL-3

SystemRequirements libjpeg

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

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

NeedsCompilation yes

Repository CRAN

Date/Publication 2022-11-29 11:12:52 UTC

R topics documented:

`readJPEG` *Read a bitmap image stored in the JPEG format*

Description

Reads an image from a JPEG file/content into a raster array.

Usage

```
readJPEG(source, native = FALSE)
```

Arguments

source	Either name of the file to read from or a raw vector representing the JPEG file content.
native	determines the image representation - if FALSE (the default) then the result is an array, if TRUE then the result is a native raster representation.

Value

If native is FALSE then an array of the dimensions height x width x channels. If there is only one channel the result is a matrix. The values are reals between 0 and 1. If native is TRUE then an object of the class nativeRaster is returned instead. The latter cannot be easily computed on but is the most efficient way to draw using rasterImage.

Most common files decompress into RGB (3 channels) or Grayscale (1 channel). Note that Grayscale images cannot be directly used in rasterImage unless native is set to TRUE because rasterImage requires RGB or RGBA format (nativeRaster is always 8-bit RGBA).

JPEG doesn't support alpha channel, you may want to use PNG instead in such situations.

Note

CMYK JPEG images saved by Adobe Photoshop may have inverted ink values due to a bug in Photoshop. Unfortunately this includes some sample CMYK images that are floating around, so beware of the source when converting the result to other color spaces. readJPEG will preserve values exactly as they are encoded in the file.

See Also

rasterImage, writeJPEG

Examples

```
# read a sample file (R logo)
img <- readJPEG(system.file("img", "Rlogo.jpg", package="jpeg"))

# read it also in native format
img.n <- readJPEG(system.file("img", "Rlogo.jpg", package="jpeg"), TRUE)

# if your R supports it, we'll plot it
if (exists("rasterImage")) { # can plot only in R 2.11.0 and higher
  plot(1:2, type='n')

  rasterImage(img, 1.2, 1.27, 1.8, 1.73)
  rasterImage(img.n, 1.5, 1.5, 1.9, 1.8)
}
```

writeJPEG *Write a bitmap image in JPEG format*

Description

Create a JPEG image from an array or matrix.

Usage

```
writeJPEG(image, target = raw(), quality = 0.7, bg = "white", color.space)
```

Arguments

image	image represented by a real matrix or array with values in the range of 0 to 1. Values outside this range will be clipped. The object must be either two-dimensional (grayscale matrix) or three dimensional array (third dimension specifying the plane) and must have either one (grayscale), two (grayscale + alpha), three (RGB) or four (RGB + alpha) planes. (For alternative image specifications see deatils)
target	Either name of the file to write to, or a binary connection, or a raw vector (raw() - the default - is good enough) indicating that the output should be a raw vector.
quality	JPEG quality - a real number between 0 (lowest) and 1 (highest) controlling the quality of the output. Lower quality produces smaller, but more lossy files.
bg	background color - used only if the input contains alpha channel since JPEG does not support storage of the alpha channel and thus the image needs to be flattened as if it was placed over the background of this color.
color.space	color space in which the image data is to be interpreted. Defaults to the "color.space" attribute of the image and NULL is interpreted as the default color space. The color space specified here must match the image array dimensions, no conversions are performed. Currently the only supported non-default color space is "CMYK" for four-channel images (which would be interpreted as "RGBA" if the color space is not specified).

Details

writeJPEG takes an image as input and compresses it into JPEG format. The image input is usually a matrix (for grayscale images - dimensions are width, height) or an array (for color and alpha images - dimensions are width, height, planes) of reals. The planes are interpreted in the sequence red, green, blue, alpha. For convenience writeJPEG allows the source to include alpha channel, but JPEG does NOT support alpha channel so it will be blended against the specified background.

Alternative representation of an image is of nativeRaster class which is an integer matrix with each entry representing one pixel in binary encoded RGBA format (as used internally by R). It can be obtained from readJPEG using native = TRUE.

Finally, writeJPEG also supports raw array containing the RGBA (or CMYK) image as bytes. The dimensions of the raw array have to be planes, width, height (because the storage is interleaved).

Currently only 4 planes (RGBA and CMYK) are supported and the processing of RGBA is equivalent to that of a native raster.

The result is either stored in a file (if `target` is a file name), send to a binary connection (if `target` is a connection) or stored in a raw vector (if `target` is a raw vector).

Value

`NULL` if the target is either a file or connection, or a raw vector containing the compressed JPEG image if the target was a raw vector.

Note

Currently `writeJPEG` only produces 8-bit, non-progressive JPEG format with no additional tags.

See Also

`readJPEG`

Examples

```
# read a sample file (R logo)
img <- readJPEG(system.file("img","Rlogo.jpg",package="jpeg"))
# write the image into a raw vector - using a low quality
r <- writeJPEG(img, raw(), quality=0.3)
# read it back again
img2 <- readJPEG(r)
# it will be slightly different since JPEG is a lossy format
# in particular at the low quality
max(abs(img - img2))
stopifnot(max(abs(img - img2)) < 0.4)

# try to write a native raster
img3 <- readJPEG(system.file("img","Rlogo.jpg",package="jpeg"), TRUE)
r2 <- writeJPEG(img3, raw())
img4 <- readJPEG(r2, TRUE)
# comparing nativeRaster values is not easy, so let's do write/read again
img5 <- readJPEG(writeJPEG(img4, raw()))
max(abs(img - img5))
stopifnot(max(abs(img - img5)) < 0.3)
```

Index