

Package: ijtiff (via r-universe)

August 5, 2026

Type Package

Title Comprehensive TIFF I/O with Full Support for 'ImageJ' TIFF Files

Version 3.3.0

Maintainer Rory Nolan <rorynoolan@gmail.com>

Description General purpose TIFF file I/O for R users. Currently the only such package with read and write support for TIFF files with floating point (real-numbered) pixels, and the only package that can correctly import TIFF files that were saved from 'ImageJ' and write TIFF files than can be correctly read by 'ImageJ' <<https://imagej.net/ij/>>. Also supports text image I/O.

License GPL-3

URL <https://docs.ropensci.org/ijtiff/>,
<https://github.com/ropensci/ijtiff>

BugReports <https://github.com/ropensci/ijtiff/issues>

Depends R (>= 4.2)

Imports checkmate (>= 1.9.3), cli, dplyr, fs (>= 1.5), graphics, grDevices, jsonlite, lubridate, magrittr (>= 1.5), methods, purrr, readr, rlang (>= 1.0), strex (>= 1.5), stringr (>= 1.4)

Suggests abind, covr, EBImage, knitr, rmarkdown, spelling, testthat (>= 3.0), tiff

VignetteBuilder knitr

Biarch TRUE

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

SystemRequirements fftw3, libtiff, libbz2, libdeflate, libjpeg, liblzma, libwebp, libzstd, X11, zlib

Config/pak/sysreqs cmake libfftw3-dev make libbz2-dev libicu-dev libjpeg-dev liblzma-dev libtiff-dev libuv1-dev libwebp-dev libzstd-dev libx11-dev zlib1g-dev

Repository https://ropensci.r-universe.dev

Date/Publication 2026-07-06 21:26:46 UTC

RemoteUrl https://github.com/ropensci/ijtiff

RemoteRef master

RemoteSha 6cac7dbe6860fb4a2b4c2f8742990b9066facc0d

Contents

as.raster.ijtiff_img	2
as_EBImage	3
count_frames	4
display	5
get_supported_tags	5
ijtiff	6
ijtiff_img	7
linescan-conversion	8
print.ijtiff_img	8
read_tags	9
read_tif	10
text-image-io	12
tif_tags_reference	13
write_tif	14
Index	17

as.raster.ijtiff_img	Convert an ijtiff_img object to a raster object for plotting
----------------------	--

Description

This function converts an [ijtiff_img](#) object to a raster object that can be used with base R graphics functions. The function extracts the first frame of the image and converts it to an RGB raster representation.

Usage

```
## S3 method for class 'ijtiff_img'
as.raster(x, ...)
```

Arguments

- x An [ijtiff_img](#) object. This should be a 4D array with dimensions representing (y, x, channel, frame).
- ... Passed to [graphics::plot.raster\(\)](#).

Details

The function performs the following operations:

- Extracts the first frame of the image
- Checks for invalid values (all NA or negative values)
- Determines the appropriate color scaling based on the image bit depth
- Creates an RGB representation using the available channels

For single-channel images, a grayscale representation is created. For RGB images (3 channels), a full-color representation is created.

Value

A raster object compatible with `graphics::plot.raster()`. The raster will represent the first frame of the input image.

Examples

```
# Read a TIFF image
img <- read_tif(system.file("img", "Rlogo.tif", package = "ijtiff"))

# Convert to raster and plot
raster_img <- as.raster(img)
plot(raster_img)
```

as_EBImage

Convert an *ijtiff_img* to an *EBImage::Image*.

Description

This is for interoperability with the the EBImage package.

Usage

```
as_EBImage(img, colormode = NULL, scale = TRUE, force = TRUE)
```

Arguments

img	An <i>ijtiff_img</i> object (or something coercible to one).
colormode	A numeric or a character string containing the color mode which can be either "Grayscale" or "Color". If not specified, a guess is made. See 'Details'.
scale	Scale values in an integer image to the range $[0, 1]$? Has no effect on floating-point images.
force	This function is designed to take <i>ijtiff_imgs</i> as input. To force any old array through this function, use <code>force = TRUE</code> , but take care to check that the result is what you'd like it to be.

Details

The guess for the colormode is made as follows: * If `img` has an attribute `color_space` with value "RGB", then `colormode` is set to "Color". * Else if `img` has 3 or 4 channels, then `colormode` is set to "Color". * Else `colormode` is set to "Grayscale".

Value

An [EBImage::Image](#).

Examples

```
if (rlang::is_installed("EBImage")) {
  img <- read_tif(system.file("img", "Rlogo.tif", package = "ijtiff"))
  str(img)
  str(as_EBImage(img))
}
```

count_frames

Count the number of frames in a TIFF file.

Description

TIFF files can hold many frames. Often this is sensible, e.g. each frame could be a time-point in a video or a slice of a z-stack.

Usage

```
count_frames(path)
```

```
frames_count(path)
```

Arguments

`path` A string, the path to the tiff file to read. `path` may also be a raw vector of TIFF bytes, read in-memory.

Details

For those familiar with TIFF files, this function counts the number of directories in a TIFF file. There is an adjustment made for some ImageJ-written TIFF files.

Value

A number, the number of frames in the TIFF file. This has an attribute `n_dirs` which holds the true number of directories in the TIFF file, making no allowance for the way ImageJ may write TIFF files.

Examples

```
count_frames(system.file("img", "Rlogo.tif", package = "ijtiff"))
```

display

Basic image display.

Description

Display an image that has been read in by `read_tif()` as it would look in 'ImageJ'. This function wraps `graphics::plot.raster()`.

Usage

```
display(img, ...)
```

Arguments

<code>img</code>	An <code>ijtiff_img</code> object.
<code>...</code>	Passed to <code>graphics::plot.raster()</code> .

Examples

```
img <- read_tif(system.file("img", "Rlogo.tif", package = "ijtiff"))
display(img)
display(img[, , 1, 1]) # first (red) channel, first frame
display(img[, , 2, 1]) # second (green) channel, first frame
display(img[, , 3, 1]) # third (blue) channel, first frame
display(img, basic = TRUE) # displays first (red) channel, first frame
```

get_supported_tags

Get supported TIFF tags

Description

Returns a named integer vector of supported TIFF tags. The names are the human-readable tag names, and the values are the corresponding tag codes.

Usage

```
get_supported_tags()
```

Value

A named integer vector of supported TIFF tags

`ijtiff`*ijtiff: TIFF I/O for ImageJ users*

Description

This is a general purpose TIFF I/O utility for R. The **tiff package** already exists for this purpose but `ijtiff` adds some functionality and overcomes some bugs therein.

Details

- `ijtiff` can write TIFF files whose pixel values are real (floating-point) numbers; `tiff` cannot.
- `ijtiff` can read and write *text images*; `tiff` cannot.
- `tiff` struggles to interpret channel information and gives cryptic errors when reading TIFF files written by the *ImageJ* software; `ijtiff` works smoothly with these images.

Author(s)

Maintainer: Rory Nolan <rorynolan@gmail.com> (**ORCID**)

Authors:

- Kent Johnson <kjohnson@akoyabio.com>

Other contributors:

- Simon Urbanek <Simon.Urbanek@r-project.org> [contributor]
- Sergi Padilla-Parra <spadilla@well.ox.ac.uk> (**ORCID**) [thesis advisor]
- Jeroen Ooms (**ORCID**) [reviewer, contributor]
- Jon Clayden (**ORCID**) [reviewer]

See Also

Useful links:

- <https://docs.ropensci.org/ijtiff/>
- <https://github.com/ropensci/ijtiff>
- Report bugs at <https://github.com/ropensci/ijtiff/issues>

ijtiff_img	ijtiff_img <i>class</i> .
------------	---------------------------

Description

A class for images which are read or to be written by the ijtiff package.

Usage

```
ijtiff_img(img, ...)

as_ijtiff_img(img, ...)
```

Arguments

img	An array representing the image. • For a single-plane, grayscale image, use a matrix <code>img[y, x]</code>. • For a multi-plane, grayscale image, use a 3-dimensional array <code>img[y, x, plane]</code>. • For a multi-channel, single-plane image, use a 4-dimensional array with a redundant 4th slot <code>img[y, x, channel,]</code> (see ijtiff_img 'Examples' for an example). • For a multi-channel, multi-plane image, use a 4-dimensional array <code>img[y, x, channel, plane]</code>.
...	Named arguments which are set as attributes.

Value

A 4 dimensional array representing an image, indexed by `img[y, x, channel, frame]`, with selected attributes.

Examples

```
img <- matrix(1:4, nrow = 2) # to be a single-channel, grayscale image
ijtiff_img(img, description = "single-channel, grayscale")
img <- array(seq_len(2^3), dim = rep(2, 3)) # 1 channel, 2 frame
ijtiff_img(img, description = "blah blah blah")
img <- array(seq_len(2^3), dim = c(2, 2, 2, 1)) # 2 channel, 1 frame
ijtiff_img(img, description = "blah blah")
img <- array(seq_len(2^4), dim = rep(2, 4)) # 2 channel, 2 frame
ijtiff_img(img, software = "R")
```

linescan-conversion *Rejig linescan images.*

Description

ijtiff has the fourth dimension of an [ijtiff_img](#) as its time dimension. However, some linescan images (images where a single line of pixels is acquired over and over) have the time dimension as the y dimension, (to avoid the need for an image stack). These functions allow one to convert this type of image into a conventional [ijtiff_img](#) (with time in the fourth dimension) and to convert back.

Usage

```
linescan_to_stack(linescan_img)
```

```
stack_to_linescan(img)
```

Arguments

linescan_img	A 4-dimensional array in which the time axis is the first axis. Dimension 4 must be 1 i.e. <code>dim(linescan_img)[4] == 1</code> .
img	A conventional ijtiff_img , to be turned into a linescan image. Dimension 1 must be 1 i.e. <code>dim(img)[1] == 1</code> .

Value

The converted image, an object of class [ijtiff_img](#).

Examples

```
linescan <- ijtiff_img(array(rep(1:4, each = 4), dim = c(4, 4, 1, 1)))
print(linescan)
stack <- linescan_to_stack(linescan)
print(stack)
linescan <- stack_to_linescan(stack)
print(linescan)
```

print.ijtiff_img *Print method for an ijtiff_img.*

Description

Print method for an [ijtiff_img](#).

Usage

```
## S3 method for class 'ijtiff_img'
print(x, ...)
```

Arguments

x An object of class [ijtiff_img](#).
 ... Not currently used.

Value

The input (invisibly).

read_tags	<i>Read TIFF tag information without actually reading the image array.</i>
-----------	--

Description

TIFF files contain metadata about images in their *TIFF tags*. This function is for reading this information without reading the actual image.

Usage

```
read_tags(path, frames = "all", translate_tags = TRUE)

tags_read(path, frames = 1)
```

Arguments

path A string, the path to the tiff file to read. path may also be a raw vector of TIFF bytes, read in-memory.
 frames Which frames do you want to read. Default all. To read the 2nd and 7th frames, use frames = c(2, 7).
 translate_tags Logical. Should the TIFF tags be translated to human-readable strings? E.g. Compression = 1 becomes Compression = "none".

Value

A list of lists.

Author(s)

Simon Urbanek, Kent Johnson, Rory Nolan.

See Also

[read_tif\(\)](#)

Examples

```
read_tags(system.file("img", "Rlogo.tif", package = "ijttiff"))
```

read_tif

Read an image stored in the TIFF format

Description

Reads an image from a TIFF file/content into a numeric array or list.

Usage

```
read_tif(
  path,
  frames = "all",
  list_safety = "error",
  msg = TRUE,
  convert = "auto"
)

tif_read(
  path,
  frames = "all",
  list_safety = "error",
  msg = TRUE,
  convert = "auto"
)
```

Arguments

path	A string, the path to the tiff file to read. path may also be a raw vector of TIFF bytes, read in-memory.
frames	Which frames do you want to read. Default all. To read the 2nd and 7th frames, use frames = c(2, 7).
list_safety	A string. This is for type safety of this function. Since returning a list is unlikely and probably unexpected, the default is to error. You can instead opt to throw a warning (list_safety = "warning") or to just return the list quietly (list_safety = "none").
msg	Print an informative message about the image being read?
convert	One of "auto" (default), "force", "never", controlling libtiff's RGBA decoder. "auto" reads every image directly and only falls back to RGBA for images the direct decoder cannot parse at all (certain tiled/planar or odd-bit-depth files that would otherwise error), preserving existing behaviour for all currently-readable images. "force" always decodes via RGBA, converting any colorspace (YCbCr, CMYK, CIELab, ...) to 8-bit RGB(A) — use this to read

such images in RGB. "never" never uses the RGBA decoder and errors on images the direct decoder cannot parse. Note: for images with frames of differing dimensions read as a list, the converted arrays keep their raw per-frame dimensions and are not reconciled.

Details

TIFF files have the capability to store multiple images, each having multiple channels. Typically, these multiple images represent the sequential frames in a time-stack or z-stack of images and hence each of these images has the same dimension. If this is the case, they are all read into a single 4-dimensional array `img` where `img` is indexed as `img[y, x, channel, frame]` (where we have `y`, `x` to comply with the conventional row, col indexing of a matrix - it means that images displayed as arrays of numbers in the R console will have the correct orientation). However, it is possible that the images in the TIFF file have varying dimensions (most people have never seen this), in which case they are read in as a list of images, where again each element of the list is a 4-dimensional array `img`, indexed as `img[y, x, channel, frame]`.

A (somewhat random) set of TIFF tags are attributed to the read image. These are `ImageDepth`, `BitsPerSample`, `SamplesPerPixel`, `SampleFormat`, `PlanarConfig`, `Compression`, `Threshholding`, `XResolution`, `YResolution`, `ResolutionUnit`, `Indexed` and `Orientation`. More tags should be added in a subsequent version of this package. You can read about TIFF tags at <https://www.awaresystems.be/imaging/tiff/tifftags.html>.

TIFF images can have a wide range of internal representations, but only the most common in image processing are supported (8-bit, 16-bit and 32-bit integer and 32-bit float samples, and 12-bit grayscale integer samples).

Value

An object of class `ijtiff_img` or a list of `ijtiff_imgs`.

Note

- 12-bit grayscale TIFFs are supported (values returned in 0..4095, assuming big-endian packing); 12-bit colour, colormap, and tiled 12-bit TIFFs are not.
- There is no standard for packing order for TIFFs beyond 8-bit so we assume big-endian packing.
- RGBA conversion (`convert = "auto"/"force"`) is unavailable for tiled separate-planar images, which therefore cannot be read at all

Author(s)

Simon Urbanek wrote most of this code for the 'tiff' package. Rory Nolan lifted it from there and changed it around a bit for this 'ijtiff' package. Credit should be directed towards Lord Urbanek.

See Also

`write_tif()`

Examples

```
img <- read_tif(system.file("img", "Rlogo.tif", package = "ijtiff"))
```

text-image-io	<i>Read/write an image array to/from disk as text file(s).</i>
---------------	--

Description

Write images (arrays) as tab-separated .txt files on disk. Each channel-frame pair gets its own file.

Usage

```
write_txt_img(img, path, rds = FALSE, msg = TRUE)
```

```
read_txt_img(path, msg = TRUE)
```

```
txt_img_write(img, path, rds = FALSE, msg = TRUE)
```

```
txt_img_read(path, msg = TRUE)
```

Arguments

img	An image, represented by a 4-dimensional array, like an ijtiff_img .
path	The name of the input/output output file(s), <i>without</i> a file extension.
rds	In addition to writing a text file, save the image as an RDS (a single R object) file?
msg	Print an informative message about the image being read?

Examples

```
img <- read_tif(system.file("img", "Rlogo.tif", package = "ijtiff"))
tmptxt <- tempfile(pattern = "img", fileext = ".txt")
write_txt_img(img, tmptxt)
tmptxt_ch1_path <- paste0(strex::str_before_last_dot(tmptxt), "_ch1.txt")
print(tmptxt_ch1_path)
txt_img <- read_txt_img(tmptxt_ch1_path)
```

tif_tags_reference	<i>TIFF tag reference.</i>
--------------------	----------------------------

Description

A dataset containing the information on all known baseline and extended TIFF tags.

Usage

```
tif_tags_reference()
```

Details

A data frame with 96 rows and 10 variables:

code_dec decimal numeric code of the TIFF tag

code_hex hexadecimal numeric code of the TIFF tag

name the name of the TIFF tag

short_description a short description of the TIFF tag

tag_type the type of TIFF tag: either "baseline" or "extended"

url the URL of the TIFF tag at <https://www.awaresystems.be>

libtiff_name the TIFF tag name in the libtiff C library

c_type the C type of the TIFF tag data in libtiff

count the number of elements in the TIFF tag data

default the default value of the data held in the TIFF tag

Source

<https://www.awaresystems.be>

Examples

```
tif_tags_reference()
```

write_tif

Write images in TIFF format

Description

Write images into a TIFF file.

Usage

```
write_tif(
    img,
    path,
    bits_per_sample = "auto",
    compression = "none",
    overwrite = FALSE,
    msg = TRUE,
    tags_to_write = NULL
)
```

```
tif_write(
    img,
    path,
    bits_per_sample = "auto",
    compression = "none",
    overwrite = FALSE,
    msg = TRUE,
    tags_to_write = NULL
)
```

Arguments

img	An array representing the image. • For a single-plane, grayscale image, use a matrix <code>img[y, x]</code>. • For a multi-plane, grayscale image, use a 3-dimensional array <code>img[y, x, plane]</code>. • For a multi-channel, single-plane image, use a 4-dimensional array with a redundant 4th slot <code>img[y, x, channel,]</code> (see ijtiff_img 'Examples' for an example). • For a multi-channel, multi-plane image, use a 4-dimensional array <code>img[y, x, channel, plane]</code>.
path	Path to the TIFF file to write to. Pass <code>path = NULL</code> to write to memory; <code>write_tif</code> then returns the TIFF file as a raw vector instead of writing to disk.
bits_per_sample	Number of bits per sample (numeric scalar). Supported values are 8, 16, and 32. The default "auto" automatically picks the smallest workable value based on the maximum element in <code>img</code>. For example, if the maximum element in <code>img</code>

	is 789, then 16-bit will be chosen because 789 is greater than $2^8 - 1$ but less than or equal to $2^{16} - 1$.
compression	A string, the desired compression algorithm. Must be one of "none", "LZW", "PackBits", "RLE", "JPEG", "deflate" or "Zip". If you want compression but don't know which one to go for, I recommend "Zip", it gives a large file size reduction and it's lossless. Note that "deflate" and "Zip" are the same thing. Avoid using "JPEG" compression in a TIFF file if you can; I've noticed it can be buggy.
overwrite	If writing the image would overwrite a file, do you want to proceed?
msg	Print an informative message about the image being written?
tags_to_write	A named list of TIFF tags to write. Tag names are case-insensitive and hyphens/underscores are ignored (e.g., "X_Resolution", "x-resolution", and "xresolution" are all equivalent). If NULL (the default), any supported TIFF tags present as attributes on <code>img</code> (e.g., from <code>read_tif()</code>) will be automatically extracted and written. If <code>tags_to_write</code> is specified, it is merged with auto-detected tags, with explicit tags taking precedence. This enables round-trip editing where metadata from a read image is preserved on re-export unless explicitly overridden. Supported tags are: • <code>xresolution</code> - Numeric value for horizontal resolution in pixels per unit • <code>yresolution</code> - Numeric value for vertical resolution in pixels per unit • <code>resolutionunit</code> - Integer: 1 (none), 2 (inch), or 3 (centimeter) • <code>orientation</code> - Integer 1-8 for image orientation • <code>xposition</code> - Numeric value for horizontal position in resolution units • <code>yposition</code> - Numeric value for vertical position in resolution units • <code>copyright</code> - Character string for copyright notice • <code>artist</code> - Character string for creator name • <code>documentname</code> - Character string for document name • <code>datetime</code> - Date/time (character, Date, or POSIXct) • <code>imagedescription</code> - Character string for image description • <code>photometric</code> - "RGB", "BlackIsZero", or "WhiteIsZero" (or integer code 0/1/2). Default is "BlackIsZero".

Value

The input `img` (invisibly), or — when `path = NULL` — a raw vector of the TIFF bytes.

Author(s)

Simon Urbanek wrote most of this code for the 'tiff' package. Rory Nolan lifted it from there and changed it around a bit for this 'ijtiff' package. Credit should be directed towards Lord Urbanek.

See Also

[read_tif\(\)](#)

Examples

```
img <- read_tif(system.file("img", "Rlogo.tif", package = "ijtiff"))
temp_dir <- tempdir()

# Basic write
write_tif(img, paste0(temp_dir, "/", "Rlogo"))

# Round-trip with automatic metadata preservation
# Metadata from read_tif is automatically preserved
img_with_metadata <- read_tif(system.file("img", "Rlogo.tif",
                                          package = "ijtiff"))
write_tif(img_with_metadata, paste0(temp_dir, "/", "Rlogo_preserved"))

# Write with additional/overridden tags
write_tif(img_with_metadata, paste0(temp_dir, "/", "Rlogo_with_tags"),
          tags_to_write = list(
            artist = "R Core Team",
            copyright = "(c) 2024",
            imagedescription = "The R logo"
          ))

img <- matrix(1:4, nrow = 2)
write_tif(img, paste0(temp_dir, "/", "tiny2x2"))
list.files(temp_dir, pattern = "tif$")
```


Index

`as.raster.ijtiff_img`, 2
`as_EBImage`, 3
`as_ijtiff_img (ijtiff_img)`, 7
`count_frames`, 4
`display`, 5
`EBImage::Image`, 3, 4
`frames_count (count_frames)`, 4
`get_supported_tags`, 5
`graphics::plot.raster()`, 2, 3, 5
`ijtiff`, 6
`ijtiff-package (ijtiff)`, 6
`ijtiff_img`, 2, 3, 5, 7, 7–9, 11, 12, 14
`linescan-conversion`, 8
`linescan_to_stack`
 (`linescan-conversion`), 8
`print.ijtiff_img`, 8
`read_tags`, 9
`read_tif`, 10
`read_tif()`, 5, 9, 15
`read_txt_img (text-image-io)`, 12
`stack_to_linescan`
 (`linescan-conversion`), 8
`tags_read (read_tags)`, 9
`text-image-io`, 12
`tif_read (read_tif)`, 10
`tif_tags_reference`, 13
`tif_write (write_tif)`, 14
`txt_img_read (text-image-io)`, 12
`txt_img_write (text-image-io)`, 12
`write_tif`, 14
`write_tif()`, 11
`write_txt_img (text-image-io)`, 12