Package: scpImaging (via r-universe)

May 26, 2026

Title Functions for handling cellenONE-based imaging data for single-cell experiments

Version 0.9.9.2

Description This package includes functions for curating, manipulating, and visualising cellenONE-based imaging data for single-cell experiments. It is designed to work with microscopy images and associated metadata produced on the cellenONE platform, which is a high-throughput imaging and handling system for single-cell analysis. It contains functions to process the initial images for downstream analysis in cellPose and cellProfiler, as well as for importing and processing the data output from these pipelines. It also contains a module for interactive visualisation of cellenONE-based images using the iSEE package for interactive single-cell experiment exploration.

Imports magick, reticulate, S4Vectors, iSEE, rmarkdown, shiny, methods, dplyr, rlang, stringr, SummarizedExperiment, ggplot2, patchwork, png, shinyWidgets, QFeatures, SingleCellExperiment

License MIT License + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Config/pak/sysreqs cmake libglpk-dev make libmagick++-dev gsfonts libicu-dev libpng-dev libuv1-dev libxml2-dev libssl-dev perl python3 zlib1g-dev

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

Date/Publication 2025-11-25 12:37:02 UTC

RemoteUrl https://github.com/emmottlab/scpImaging

RemoteRef HEAD

RemoteSha ca948ba151adb8b1057c70961521a062b695b29d

Contents

`addCellProfilerAttributes` *'addCellProfilerAttributes()': Add Attributes from a Second Data Frame Based on Keys*

Description

Merges attributes (columns) from a second data frame (`df2`) into a primary data frame (`df1`) based on specified key columns. This function performs a left join, keeping all rows from `df1`. It provides options for handling cases where keys in `df2` represent multiple measurements (multiplets).

Usage

```
addCellProfilerAttributes(
  df1,
  df2,
  key1_col = "Cropped",
  key2_col = "FileName_Image",
  multiplet_handling = c("voidmultiplets", "takemaxarea"),
  area_col = "AreaShape_Area"
)
```

Arguments

`df1`	The primary data frame (data.frame or tibble). All rows from this data frame will be kept.
`df2`	The secondary data frame (data.frame or tibble) containing the attributes to add.
`key1_col`	The name of the key column in `df1` (character string). Defaults to `"Cropped"`.
`key2_col`	The name of the key column in `df2` (character string). Defaults to `"FileName_Image"`.

multiplet_handling
: Method to handle multiple rows for the same key in df2. Must be one of "voidmultiplets" or "takemaxarea" (character string). Defaults to "voidmultiplets".

area_col
: The name of the column in df2 containing the area values, required only if multiplet_handling = "takemaxarea" (character string). Defaults to "AreaShape_Area".

Details

The function uses key columns key1_col from df1 and key2_col from df2 to match rows. All rows from df1 are preserved.

Handling of multiple rows for the same key (multiplets) in df2:

- voidmultiplets (default): If a key in key2_col appears in multiple rows of df2, the corresponding rows in the merged output will have NA values for all columns originating from df2.
- takemaxarea: If a key in key2_col appears in multiple rows of df2, only the data from the single df2 row having the maximum value in the area_col column will be added to df1, i.e. the largest cell. If the values are identical, will take the first encountered row among those with the maximum area.

Value

A data frame containing all rows and columns from df1, augmented with columns from df2 based on the matched keys and the specified multiplet_handling strategy. Columns added from df2 will exclude the key column (key2_col).

Examples

```
# Please see vignette
```

---

addCellProfilerAttributesSCE

'addCellProfilerAttributesSCE()': Add CellProfiler Attributes to colData of Bioconductor Objects

---

Description

A wrapper function for addCellProfilerAttributes that operates on the colData of SummarizedExperiment, SingleCellExperiment, or QFeatures objects.

Usage

```
addCellProfilerAttributesSCE(
  sce_obj,
  df2,
  key1_col = "Cropped",
  key2_col = "FileName_Image",
  multiplet_handling = c("voidmultiplets", "takemaxarea"),
  area_col = "AreaShape_Area"
)
```

Arguments

sce_obj	A SummarizedExperiment, SingleCellExperiment, or QFeatures object.
df2	The secondary data frame (data.frame or tibble) containing the attributes to add. Passed directly to addCellProfilerAttributes.
key1_col	The name of the key column *in the* colData *of* sce_obj (character string). Defaults to "Cropped". Passed to addCellProfilerAttributes.
key2_col	The name of the key column in df2 (character string). Defaults to "FileName_Image". Passed to addCellProfilerAttributes.
multiplet_handling	Method to handle multiple rows for the same key in df2. Must be one of "voidmultiplets" or "takemaxarea" (character string). Defaults to "voidmultiplets". Passed to addCellProfilerAttributes.
area_col	The name of the column in df2 containing the area values, required only if multiplet_handling = "takemaxarea" (character string). Defaults to "AreaShape_Area". Passed to addCellProfilerAttributes.

Details

This function extracts the colData from the input object (sce_obj), converts it to a standard data.frame, and passes it as df1 to the addCellProfilerAttributes function along with df2 and other parameters. The resulting merged data frame is then converted back to a DataFrame and used to replace the original colData in the input object.

The row names of the colData are preserved throughout the process. For QFeatures objects, this function modifies the primary (top-level) colData.

See ?addCellProfilerAttributes for details on the merging logic, multiplet_handling options, and other parameters.

Value

An object of the same class as sce_obj with its colData updated to include the merged attributes from df2.

Examples

```
# --- Create Sample SummarizedExperiment ---
library(SummarizedExperiment)
```

```
library(S4Vectors)
counts <- matrix(rpois(100, lambda = 10), ncol=10, nrow=10)
rownames(counts) <- paste0("Gene", 1:10)
colnames(counts) <- paste0("Cell", 1:10)

sample_coldata <- DataFrame(
   SampleID = paste0("S", 1:10),
   Treatment = rep(c("A", "B"), 5),
   # Key column matching df1's perspective in addCellProfilerAttributes
   Cropped_Path = paste0("path/", letters[1:10]),
   row.names = colnames(counts) # Ensure row names match assay colnames
)

sce <- SummarizedExperiment(assays=list(counts=counts),
                            colData=sample_coldata)

print("Original colData:")
print(colData(sce))

# --- Create Sample df2 (attributes to add) ---
df2_attribs <- data.frame(
  FileName_Image = c("path/a", "path/b", "path/b", "path/c", "path/f", "path/g"),
  QC_metric = rnorm(6, mean=100),
  AreaShape_Area = c(50, 60, 55, 70, 80, 90),
  Batch = rep(c("X", "Y"), 3)
)
print("Attributes to add (df2):")
print(df2_attribs)
```

`CellenONEPlot` *Create an Instance of a CellenONEPlot Panel*

Description

This function is the constructor for creating `CellenONEPlot-class` objects, which represent an iSEE panel for visualizing CellenONE microscopy images.

Usage

```
CellenONEPlot(...)
```

Arguments

`...`	Arguments passed to the `initialize` method of the `CellenONEPlot-class`. These can include initial values for any of the slots defined in `CellenONEPlot-class` or its parent classes, such as `ImageFile_Path`, `Background_Path`, `ImageDir`, etc.

Value

A CellenONEPlot object.

Author(s)

Ed Emmott

See Also

CellenONEPlot-class for details on the class slots and methods.

Examples

```
# Basic constructor
# CellenONEPlot()

# Constructor with custom slot values (if 'se' is a SummarizedExperiment)
# CellenONEPlot(ImageDir = "my_image_directory/", YAxisSampleName = colnames(se)[1])
```

CellenONEPlot-class *The CellenONEPlot Class*

Description

The CellenONEPlot class displays CellenONE and scpImaging pipeline-derived micrographs dynamically, as relevant entries in a single-cell experiment are selected through an iSEE app.

Usage

```
## S4 method for signature 'CellenONEPlot'
.refineParameters(x, se)

## S4 method for signature 'CellenONEPlot'
.singleSelectionSlots(x)

## S4 method for signature 'CellenONEPlot'
.definePanelTour(x)
```

Arguments

x	A CellenONEPlot object.
se	A SummarizedExperiment object containing the data.

Value

A modified CellenONEPlot object with updated parameters, or NULL if refinement is not possible (e.g., no samples in se).

A list specifying the single selection slots, adding the Y-axis sample selection configuration to those inherited from the parent class.

Methods (by generic)

- `.definePanelTour(CellenONEPlot)`: Defines the interactive tour for this panel.

Functions

- `.refineParameters(CellenONEPlot)`: Method to refine parameters after initialization or upon changes to the `SummarizedExperiment` object. It ensures that a valid sample name is selected from the available samples in `colData(se)`.
- `.singleSelectionSlots(CellenONEPlot)`: Method to define the slots involved in single selections for this panel. This allows the panel to react to or transmit single selections (e.g., selected sample name for the Y-axis).

Slot overview

The following slots control the data retrieval and display:

- `ImageFile_Path`, character. The column name in `colData(se)` for parent image filenames. Default: `"ImageFile"`.
- `Background_Path`, character. The column name in `colData(se)` for background image filenames. Default: `"Background"`.
- `OverlayParent_Path`, character. The column name in `colData(se)` for overlayed parent image filenames. Default: `"CP_Overlay_Parent"`.
- `Cropped_Path`, character. The column name in `colData(se)` for cropped image filenames. Default: `"Cropped"`.
- `Overlay_Path`, character. The column name in `colData(se)` for cropped and overlayed image filenames. Default: `"CP_Overlay"`.
- `Mask_Path`, character. The column name in `colData(se)` for mask filenames. Default: `"CP_Mask"`.
- `ImageDir`, character. The directory path prefixed to all image filenames. Default: `"www"`.

In addition, this class inherits all slots from its parent Panel class.

Constructor

`CellenONEPlot(...)` creates an instance of a CellenONEPlot class, where any slot and its value can be passed to `...` as a named argument.

Author(s)

Ed Emmott

See Also

SampleAssayPlot

cleanFilenames *'cleanFilenames()': Clean Hyperlink Formatting from DataFrame Columns*

Description

This function removes specified prefix and suffix patterns, typically hyperlink formatting (e.g., from Excel), from character columns within a data frame.

Usage

```
cleanFilenames(
  data,
  column_names,
  prefix_pattern = "^=HYPERLINK\\(\"",
  suffix_pattern = "\"\\)$"
)
```

Arguments

`data`	A `data.frame` containing the columns to be cleaned.
`column_names`	A character vector specifying the names of the columns in `data` that need to be cleaned.
`prefix_pattern`	A character string containing a regular expression for the prefix to be removed from the start of the strings in the specified columns. Defaults to `'^=HYPERLINK\\("'`.
`suffix_pattern`	A character string containing a regular expression for the suffix to be removed from the end of the strings in the specified columns. Defaults to `'"\\)$'`.

Details

The function iterates through each column specified in `column_names`. For each column:

1. It first checks if the column is of character type. If not, a warning is issued for that column, and it is skipped (returned as is).
2. If it is a character column, it sequentially applies two `gsub` operations:
   - Removes the `prefix_pattern` from the beginning of each string.
   - Removes the `suffix_pattern` from the end of each string (from the result of the prefix removal).

The default patterns are designed to clean strings like `"=HYPERLINK(\"path/to/file.jpg\")"` into `"path/to/file.jpg"`.

Value

A `data.frame` with the specified formatting removed from the target columns. Columns not specified or not of character type (after a warning) will remain unchanged.

Examples

```
if (requireNamespace("dplyr", quietly = TRUE)) {
  cDat_example <- data.frame(
    ID = 1:3,
    ImageFile = c('=HYPERLINK("image1.png")',
                  'plain_file.jpg',
                  '=HYPERLINK("image2.tif")'),
    Background = c('=HYPERLINK("bg1.png")',
                   '=HYPERLINK("bg2.jpg")',
                   'no_hyperlink_here'),
    Notes = c("Note1", "=HYPERLINK(\"doc1.pdf\")", "Note3"),
    NumericCol = c(10,20,30),
    FactorCol = factor(c("A", "B", "A")),
    stringsAsFactors = FALSE
  )

  # Clean 'ImageFile' and 'Background' columns with default patterns
  cleaned_data1 <- cleanFilenames(cDat_example,
                                  column_names = c("ImageFile", "Background"))
  print("Cleaned ImageFile and Background:")
  print(cleaned_data1)

  # Clean 'Notes' column with default patterns
  cleaned_data_notes <- cleanFilenames(cDat_example, column_names = "Notes")
  print("Cleaned Notes:")
  print(cleaned_data_notes)

}
#' @importFrom dplyr mutate across all_of
```

---

countCells *'countCells()':Count Unique Masks in PNG Segmentation Masks*

Description

Processes PNG segmentation masks generated from cellpose .npy files by counting the number of unique positive integer values. These values correspond to individual segmented objects (e.g., cells), while background pixels (value 0) are ignored. The function can handle input specified as a directory containing masks, an explicit list of mask file paths, or a directory and a filename suffix.

Usage

```
countCells(
  path = NULL,
  filelist = NULL,
  suffix = "_cp_masks.png",
  expected_bit_depth = 8
)
```

Arguments

path	A character string specifying the path to the directory containing the PNG mask files. Mutually exclusive with filelist. Defaults to NULL.
filelist	A character vector of full file paths to the PNG mask files. Mutually exclusive with path. Defaults to NULL.
suffix	A character string specifying the filename suffix (pattern) to match when path is provided. The pattern is matched at the end of the filename. Defaults to _cp_masks.png. Ignored if filelist is provided.
expected_bit_depth	A numeric value indicating the expected bit depth of the original PNG mask files (typically 8 or 16). This is used to interpret the raw pixel data correctly. Defaults to 8.

Details

This function reads PNG files using magick::image_read. It then uses magick::image_data to extract the raw pixel data for the grayscale channel.

Because magick::image_info may not reliably return the bit depth for all PNG files, this function uses the expected_bit_depth argument to interpret the pixel data (bytes):

- If expected_bit_depth is 8, raw bytes directly represent integer values (0-255).
- If expected_bit_depth is 16, pairs of raw bytes are combined to reconstruct the 16-bit integer values (0-65535), assuming big-endian byte order.

The function assumes that the input PNGs are grayscale integer images where different masks are represented by unique positive integers and the background is 0. If non-grayscale images are encountered, a warning is issued, and the function attempts to process the grayscale information.

The function includes error handling: if a file cannot be read or processed (e.g., corrupted file, non-PNG format, unsupported expected bit depth), a warning is issued, and NA is returned for that file's count.

Provide *either* path (optionally with suffix) *or* filelist, but not both.

Value

A dataframe with two columns:

FileName	Character, the base name of the processed PNG file.
CP_CellCount	Integer, the number of unique positive integer mask values found in the image. Contains NA if the file could not be processed.

Examples

```
## Not run:
# Example Setup: Create dummy PNG mask files using the 'magick' package
temp_dir <- tempdir()
# Mask 1: 2 objects (IDs 1, 2)
mask1_data <- matrix(c(0, 0, 1, 1, 0, 2, 2, 0, 0), nrow = 3, byrow = TRUE)
# Mask 2: 4 objects (IDs 1, 2, 3, 4)
```

```
mask2_data <- matrix(c(1, 1, 1, 0, 0, 0, 2, 3, 4), nrow = 3, byrow = TRUE)
# Mask 3: 3 objects (IDs 10, 20, 300) - requires 16-bit
mask3_data_16bit <- matrix(c(0, 10, 20, 300), nrow = 2, byrow = TRUE)

# Create magick images from matrices (scale to 0-1 for image_read)
# Save as 16-bit PNGs to handle mask3
max_val_16bit <- 65535
img1_magick <- magick::image_read(mask1_data / max_val_16bit)
img2_magick <- magick::image_read(mask2_data / max_val_16bit)
img3_magick <- magick::image_read(mask3_data_16bit / max_val_16bit)

# Set depth to 16-bit before writing
img1_magick <- magick::image_depth(img1_magick, 16)
img2_magick <- magick::image_depth(img2_magick, 16)
img3_magick <- magick::image_depth(img3_magick, 16)

# Write PNGs using magick
magick::image_write(img1_magick,
                    path = file.path(temp_dir, "imageA_masks.png"),
                    format = "png")
magick::image_write(img2_magick,
                    path = file.path(temp_dir, "imageB_masks.png"),
                    format = "png")
magick::image_write(img3_magick,
                    path = file.path(temp_dir, "imageC_16bit_masks.png"),
                    format = "png")

# Create a non-png file to test error handling
writeLines("not a png", file.path(temp_dir, "not_a_png.txt"))

# --- Usage Examples ---

# 1. Using path and default suffix (".png") and default depth (16)
counts_path_default <- countCells(path = temp_dir)
print(counts_path_default)
# Expected output: Counts for A, B, C (2, 4, 3 respectively)

# 2. Using path and specific suffix, explicitly setting 16-bit depth
counts_path_suffix <- countCells(path = temp_dir,
                                          suffix = "_masks.png",
                                          expected_bit_depth = 16)
print(counts_path_suffix)
# Expected output: Counts for A, B, C (2, 4, 3 respectively)

# 3. Using filelist (including non-png and non-existent files to show NA)
file_paths <- c(
  file.path(temp_dir, "imageA_masks.png"),
  file.path(temp_dir, "imageB_masks.png"),
  file.path(temp_dir, "imageC_16bit_masks.png"),
  file.path(temp_dir, "non_existent_file.png"), # Test non-existent file
  file.path(temp_dir, "not_a_png.txt")          # Test invalid file
)
counts_filelist <- countCells(filelist = file_paths,
```

```
                                        expected_bit_depth = 16)
print(counts_filelist)
# Expected output: Counts for A, B, C, and NA for the last two with warnings.

# 4. Example assuming masks were 8-bit (will misinterpret C)
#    (Need to recreate test files as 8-bit for this to be accurate)
# counts_8bit <- countCells(path = temp_dir,
#                                     suffix = "_masks.png",
#                                     expected_bit_depth = 8)
# print(counts_8bit)

# Clean up temporary files
unlink(temp_dir, recursive = TRUE)

## End(Not run)
```

`cropImages` *'cropImages()': Generate Cropped Images from Image Paths and Coordinates*

Description

Reads images specified in a data frame, crops them based on a user-defined offset centred on user-provided pixel coordinates, and saves the cropped images to a specified output directory. Allows specifying a base input directory if image paths in the data frame are relative filenames. Output filenames are derived from the input `ImageFile` names, with options to add a prefix and replace the file extension with a custom suffix. For the intended use case, the dataframe represents the cellenONE cell sorting output file containing cell metadata and image names, the images are the images recorded for each sorted cell, and the XY coordinates represent the location of the cell in the image. The function crops a square region surrounding the cell of interest, with the size of the square determined by the pixel offset.

Usage

```
cropImages(
  df,
  pixel_offset,
  output_dir,
  input_dir = NULL,
  output_prefix = "cropped_",
  output_suffix = NULL,
  create_dir = FALSE
)
```

Arguments

`df`	A data frame containing cellenONE-derived image metadata. For example derived from a cell sorting file generated by the cellenONE, or a sampleannotation

file generated by scpAnnotator. **Required.** Must include columns: `ImageFile` (character, full path to the input image or filename relative to `input_dir` if provided), `X` (numeric, x-coordinate of the center point of the cell for cropping, as identified by the cellenONE), `Y` (numeric, y-coordinate of the center point of the cell for cropping, as identified by the cellenONE).

`pixel_offset` An integer specifying the number of pixels to offset from the center coordinates (`X`, `Y`) in each direction (up, down, left, right) to define the cropping box. **Required.** The resulting crop dimensions will be (2 * $pixel_offset$) x (2 * $pixel_offset$). Must be a positive integer. A pixel offset of 37, will create a 75 pixel crop (37 + centre pixel, + 37).

`output_dir` A character string specifying the path to the directory where cropped images should be saved. **Required.**

`input_dir` An optional character string specifying a base directory for input images. If provided, paths in `df$ImageFile` are treated as relative to this directory (e.g., if `input_dir` is "/path/to/images" and `df$ImageFile` contains "img1.png", the function will look for "/path/to/images/img1.png"). If `NULL` (the default), `df$ImageFile` must contain full paths or paths relative to the current working directory.

`output_prefix` A character string to add to the beginning of the output filenames. Defaults to ‘"cropped_"’. Optional. **Strongly recommend** leaving unchanged for consistency.

`output_suffix` An optional character string used to replace the original file extension in the output filename. For example, if `output_suffix` is `"_cropped.png"`, an input file `image1.tif` will result in an output file named `image1_cropped.png` (plus any `output_prefix`). If the original filename has no extension, the suffix is simply appended. If `NULL` or `""` (the default), the original filename (including its extension) from `ImageFile` is used as the base for the output filename. **Strongly recommend** leaving unchanged for consistency.

`create_dir` A logical value indicating whether the `output_dir` should be created if it does not exist. Defaults to `FALSE`. If `TRUE`, the function will attempt to create the directory recursively. If `FALSE` and the directory does not exist, the function will stop with an error. Optional.

Value

Returns `NULL`, so does not create an object in the R environment. The function's primary effect is saving cropped image files to the specified `output_dir`.

`generateImageColumns` *generateImageColumns(): Generate Derived Image File Columns*

Description

Takes an input data frame and adds new columns containing derived filenames based on a specified source column (e.g., cropped images, masks, overlays) .

Usage

```
generateImageColumns(data, source_column = "ImageFile", column_configs = NULL)
```

Arguments

`data`	A `data.frame` or `tibble` containing the source image file information.
`source_column`	Character string. The name of the column in `data` that contains the source filenames (e.g., image paths). Defaults to "ImageFile".
`column_configs`	An optional named list to override or add custom column configurations. Each element of the list should be named (e.g., "Cropped", "CP_Mask") and contain a sub-list with elements `col_name` (character, the desired name for the new column), `prefix` (character, prefix to add, can be `NA` or empty string for none), and `suffix` (character, suffix rule, can be `NA` or empty string for none). Suffix rules starting with "replace " (e.g., "replace.png with _mask.png") trigger string replacement; otherwise, the suffix is appended before the original extension. Defaults to `NULL`, using the internal default configurations.

Details

The function iterates through a set of predefined or user-provided configurations to generate new columns. Each configuration specifies the new column's name, an optional prefix to prepend to the source filename's base name (preserving path and extension), and an optional suffix rule applied before the original extension. Used to generate the processed filenames in scpImaging workflows.

Default Configurations:

- **Cropped**: Adds column "Cropped" with prefix "cropped_" applied to the base filename.
- **CP_Mask**: Adds column "CP_Mask" with prefix "cropped_" applied to the base filename and replaces the ".png" extension/suffix within the entire string with "_cp_masks.png".
- **CP_Overlay**: Adds column "CP_Overlay" with prefix "cropped_" applied to the base filename and replaces the ".png" extension/suffix within the entire string with "_cp_masks_overlay.png".
- **CP_Overlay_Parent**: Adds column "CP_Overlay_Parent" appending the suffix "_parent_overlayed" before the original file extension.

The `column_configs` argument allows overriding these defaults. The names of the list provided in `column_configs` should match the keys of the default configurations (e.g., "Cropped", "CP_Mask") to override them. If a configuration name in `column_configs` does not match a default, it defines a new derived column. Recommend **not** overriding the defaults.

Prefix rules add the specified string after any directory path but before the base filename.

Suffix rules are applied as follows:

- If the `suffix` string starts with "replace ", the function attempts to perform a string replacement on the *entire* generated string (after prefixing). The format expected is "replace <old_string> with <new_string>". For example, "replace.png with _mask.png". The replacement uses fixed string matching, not regular expressions.
- Otherwise, the `suffix` string is appended to the filename's base name (after the prefix, if any) but *before* the original file extension.

If the source filename is NA, the resulting derived filenames will also be NA. If a "replace" suffix rule is specified but the <old_string> is not found in a given source filename, the replacement does not occur for that filename, and a single warning summarizing all mismatches may be issued at the end. Generated column names that clash with existing columns in data will overwrite them, issuing a warning.

Value

The input data frame (tibble) with the newly generated columns added.

Examples

```
# Create a sample data frame
df <- data.frame(
  SampleID = 1:3,
  ImageFile = c(
    "path/subpath/458_Printed_T1_F1_R62_C19_(A-1)_Trans_samplename_Run.png",
    "another_image_file.tif",
    NA
  ),
  stringsAsFactors = FALSE
)

# Basic usage with defaults
result_default <- generateImageColumns(df)
print(result_default)

# Example with a different source column
df2 <- data.frame(
  ID = 1,
  OriginalPath = "path/to/image.png",
  stringsAsFactors = FALSE
)
result_source <- generateImageColumns(df2, source_column = "OriginalPath")
print(result_source)

# Example overriding defaults and adding a new column
custom_configs <- list(
  Cropped = list(col_name = "MyCropped", prefix = "mycrop_", suffix = NA),
 CP_Mask = list(col_name = "MaskFile", prefix = NA, suffix = "replace.png with _MASK.tif"),
  NewCol = list(col_name = "ExtraInfo", prefix = "info_", suffix = "_extra") # New column
)
result_custom <- generateImageColumns(df, column_configs = custom_configs)
print(result_custom)

# Example with suffix replacement mismatch (no ".png" in source)
df_mismatch <- data.frame(
  ImageFile = "image_no_png_suffix.jpeg",
  stringsAsFactors = FALSE
)
# CP_Mask and CP_Overlay rules won't replace ".png" and may trigger a warning
result_mismatch <- generateImageColumns(df_mismatch)
```

```
print(result_mismatch)
```

`generateImageColumnsSCE` *'generateImageColumnsSCE():' Apply generateImageColumns to Bioconductor Object colData*

Description

A wrapper function to apply `generateImageColumns` (a function that processes data.frames) to the `colData` of a `SummarizedExperiment` (which includes `SingleCellExperiment`) or a `QFeatures` object. It extracts the `colData`, processes it, and then updates the object's `colData` with the new columns.

Usage

```
generateImageColumnsSCE(x, ...)

## S4 method for signature 'SummarizedExperiment'
generateImageColumnsSCE(x, ...)

## S4 method for signature 'QFeatures'
generateImageColumnsSCE(x, ...)
```

Arguments

`x`	A `SummarizedExperiment`, `SingleCellExperiment`, or `QFeatures` object.
`...`	Arguments to be passed to the underlying `generateImageColumns` function that operates on data.frames. This typically includes `source_column` and `column_configs`.

Details

This function serves as a wrapper. The method for `SummarizedExperiment` also handles `SingleCellExperiment` objects due to class inheritance. For `QFeatures` objects, this function modifies the primary (global) `colData` of the `QFeatures` object.

Refer to the documentation of the original `generateImageColumns` function for details on its arguments and behavior when processing the data.frame.

Value

The input object (`x`) with its `colData` updated to include the new columns.

See Also

`generateImageColumns` (for the underlying data.frame implementation)

Examples

```
# --- Prerequisite: Define or load your original generateImageColumns function ---
# generateImageColumns <- function(data, source_column = "ImageFile", ...) {
#   # ... (your data.frame processing logic)
#   data$newCol <- paste0("processed_", data[[source_column]])
#   return(data)
# }

# --- Example for SummarizedExperiment (and by extension SingleCellExperiment) ---
library(SummarizedExperiment)
counts <- matrix(rnorm(40), ncol = 4)
rownames(counts) <- paste0("gene", 1:10)
colnames(counts) <- paste0("sample", 1:4)
sample_df <- S4Vectors::DataFrame(
  ImageFile = c("s1.png", "s2.png", "s3.png", "s4.png"),
  row.names = colnames(counts)
)
se <- SummarizedExperiment(assays = list(counts = counts), colData = sample_df)

# Assuming generateImageColumns and generateImageColumnsSCE are defined:
# print(colData(se))
# se_modified <- generateImageColumnsSCE(se, source_column = "ImageFile")
# print(colData(se_modified))

# If sce is a SingleCellExperiment, it would work the same:
# library(SingleCellExperiment)
# sce <- as(se, "SingleCellExperiment")
# sce_modified <- generateImageColumnsSCE(sce, source_column = "ImageFile")
# print(colData(sce_modified))

# --- Example for QFeatures ---
library(QFeatures)
qf_coldata <- S4Vectors::DataFrame(
  GlobalImageFile = c("q_s1.jpg", "q_s2.jpg"),
  Batch = c(1,2),
  row.names = c("qSample1", "qSample2")
)
# Creating a minimal QFeatures object for example
qf <- QFeatures(colData = qf_coldata) # No assays needed for this colData example

# print(colData(qf))
# qf_modified <- generateImageColumnsSCE(qf, source_column = "GlobalImageFile")
# print(colData(qf_modified))
```

---

npyMasksToPng *'npyMasksToPng()' converts Cellpose.npy Masks to PNG Images*

Description

Reads.npy segmentation files #' extracts the mask data, generates a pseudo-colored representation, and saves the masks as PNG images using the magick package.

Usage

```
npyMasksToPng(
  input_dir,
  output_dir = NULL,
  input_suffix = "_seg\\.npy$",
  output_suffix = "_cp_masks.png"
)
```

Arguments

input_dir	Character string. Path to the directory containing.npy files.
output_dir	Optional character string. Path to the directory where PNG files will be saved. If NULL (default), PNG files are saved in input_dir. The directory will be created if it doesn't exist.
input_suffix	Optional character string. A regular expression specifying the ending pattern of input files to process. Defaults to "_seg\\.npy$".
output_suffix	Optional character string. Suffix to use for the output PNG files (including the '.png' extension). Defaults to "_cp_masks.png".

Details

This function requires the 'reticulate' and 'magick' packages. It also depends on a Python environment accessible by 'reticulate' with the 'numpy' library installed to access the .npy files. It reads the .npy files and converts to .png images, where the background is black, and each cell mask has a unique colour.

Value

Invisibly returns a list containing vectors of successfully processed input file paths (success) and a list of errors encountered (errors), where names are input file paths and values are error messages. Primarily called for its side effect of writing PNG files.

Examples

```
# Setup: Create temporary directories and a dummy.npy file
temp_in_dir <- tempfile("input_dir_")
temp_out_dir <- tempfile("output_dir_")
dir.create(temp_in_dir)
dir.create(temp_out_dir)

# Create a dummy mask array (e.g., 3x4 with segments 1, 2, 3)
dummy_masks <- array(as.integer(c(0,1,1,0, 2,2,0,2, 0,3,3,0)), dim = c(3, 4))
# Create the dictionary structure expected within the.npy
dummy_data_to_save <- reticulate::dict("masks" = dummy_masks)
```

```
# Save the dummy data to a.npy file using numpy via reticulate
np <- reticulate::import("numpy")
npy_path <- file.path(temp_in_dir, "example_seg.npy")
tryCatch({
  np$save(npy_path, dummy_data_to_save)

  # Run the conversion function
  results <- npyMasksToPng(input_dir = temp_in_dir, output_dir = temp_out_dir)

  # Check results (output file should exist)
  print(list.files(temp_out_dir))
  print(results)

}, error = function(e) {
  message("Example skipped: numpy unavailable or failed to save file.")
})

# Cleanup
unlink(temp_in_dir, recursive = TRUE, force = TRUE)
unlink(temp_out_dir, recursive = TRUE, force = TRUE)
```

---

overlayMask *'overlayMask()': Overlay Segmentation Mask on Image (Single File or Directory)*

Description

Reads a microscopy image and a corresponding segmentation mask (or directories containing multiple images/masks), then creates an output image (or images) showing the mask as either outlines or a semi-transparent overlay on the original image. Uses the 'magick' package.

Usage

```
overlayMask(
  image_input,
  mask_input,
  output_target,
  mode = c("outline", "overlay"),
  image_suffix = NULL,
  mask_suffix = "_cp_masks.png",
  output_suffix = "_cp_masks_overlay.png",
  outline_col = "cyan",
  outline_lwd = 2,
  overlay_col = "yellow",
  overlay_alpha = 0.4
)
```

Arguments

image_input	Path to the original microscopy image file OR a directory containing multiple image files.
mask_input	Path to the segmentation mask image file OR a directory containing multiple mask files. If image_input is a directory, mask_input can be the same directory or a separate one.
output_target	Path to save the resulting composite image file OR a directory where multiple output images will be saved. If inputs are directories, this must be a directory path.
mode	Character string: "outline" or "overlay". Determines the visualization style. Default is "outline".
image_suffix	Character string or NULL. Suffix used to identify image files when image_input is a directory. Required if image_input and mask_input point to the same directory. Example: "_Run.png". If NULL and separate directories are used, attempts to match common image extensions. Default is NULL.
mask_suffix	Character string. Suffix used to identify mask files when mask_input (or image_input) is a directory. Default is "_cp_masks.png".
output_suffix	Character string. Suffix used for the generated output files. The mask file's mask_suffix will be replaced with this. Default is "_cp_masks_overlay.png". **Strongly recommend** not changing.
outline_col	Color for the outlines (used if mode="outline"). Can be any format recognized by magick. Default is "cyan".
outline_lwd	Approximate line width (thickness) for outlines in pixels (used if mode="outline"). Default is 2.
overlay_col	Color for the overlay (used if mode="overlay"). Can be any format recognized by magick. Default is "yellow".
overlay_alpha	Alpha transparency for the overlay (used if mode="overlay"). A numeric value between 0 (fully transparent) and 1 (fully opaque). Default is 0.4.

Value

Invisibly returns a list of paths to the successfully created output files.

Examples

```
## Not run:
# --- Setup Dummy Files/Dirs ---
if (requireNamespace("magick", quietly = TRUE)) {
# Create temp directories
temp_dir <- tempdir()
img_dir <- file.path(temp_dir, "images")
mask_dir <- file.path(temp_dir, "masks")
out_dir <- file.path(temp_dir, "output")
dir.create(img_dir, showWarnings = FALSE)
dir.create(mask_dir, showWarnings = FALSE)
dir.create(out_dir, showWarnings = FALSE)
```

```
# Create dummy image/mask files (1 pair)
img_base1 <- "cell_image_A1"
img_suffix <- "_microscopy.png"
mask_suffix <- "_seg_mask.png"
img_path1 <- file.path(img_dir, paste0(img_base1, img_suffix))
mask_path1 <- file.path(mask_dir, paste0(img_base1, mask_suffix))

img <- magick::image_blank(width=50, height=50, color="grey")
mask <- magick::image_blank(width=50, height=50, color="black")
mask <- magick::image_draw(mask)
graphics::symbols(25, 25, circles=10, inches=FALSE, add=TRUE, fg="white", bg="white")
grDevices::dev.off()
magick::image_write(img, img_path1)
magick::image_write(mask, mask_path1)

# Create an image file without a corresponding mask
img_base_unmatched <- "unmatched_image"
img_path_unmatched <- file.path(img_dir, paste0(img_base_unmatched, img_suffix))
magick::image_write(img, img_path_unmatched)

# Create a mask file without a corresponding image
mask_base_unmatched <- "unmatched_mask"
mask_path_unmatched <- file.path(mask_dir, paste0(mask_base_unmatched, mask_suffix))
magick::image_write(mask, mask_path_unmatched)

# --- Example 1: Single File Mode (should be quiet) ---
out_single_file <- file.path(out_dir, "single_overlay.png")
overlayMask(img_path1, mask_path1, out_single_file, mode = "overlay")

# --- Example 2: Directory Mode (should warn about unmatched files) ---
overlayMask(image_input = img_dir,
mask_input = mask_dir,
output_target = out_dir,
mode = "outline",
image_suffix = img_suffix, # Specify suffixes
mask_suffix = mask_suffix,
output_suffix = "_outline_overlay.png",
outline_col = "red")

# --- Clean up ---
unlink(img_dir, recursive = TRUE)
unlink(mask_dir, recursive = TRUE)
unlink(out_dir, recursive = TRUE)
} else {
message("Magick package not installed. Examples cannot run.")
}

## End(Not run)
```

`overlayMaskOnParent` *'overlayMaskOnParent()': Overlay Segmentation Masks onto Parent Images at Specific Coordinates*

Description

Reads parent microscopy images, corresponding segmentation masks (which match cropped regions), and a dataframe specifying the crop coordinates (e.g. a cellenONE cell sort file, or scpAnnotator file). Creates output images showing the masks overlaid onto the parent images at the correct locations, either as outlines or semi-transparent overlays. Processes images in batches based on the parent image identifier found in the coordinate dataframe.

Usage

```
overlayMaskOnParent(
  parent_image_input,
  mask_input,
  coord_df,
  output_target,
  mode = c("outline", "overlay"),
  parent_image_suffix = "_Run.png",
  mask_suffix = "_cp_masks.png",
  output_suffix = "_Run_parent_overlayed.png",
  outline_col = "cyan",
  outline_lwd = 2,
  overlay_col = "yellow",
  overlay_alpha = 0.4
)
```

Arguments

`parent_image_input`
: Path to a directory containing the parent image files.

`mask_input`
: Path to a directory containing the segmentation mask files (corresponding to cropped regions). Assumes mask names follow the pattern cropped_ ... _cp_masks.png.

`coord_df`
: An R dataframe containing the mapping information. Must include columns: 'ImageFile' (base name OR full filename used for parent files), 'X' (center x-coordinate of crop in parent), 'Y' (center y-coordinate of crop in parent). This is produced as part of the cellenONE output, or can use an scpAnnotator file.

`output_target`
: Path to a directory where the composite parent images (with overlays) will be saved.

`mode`
: Character string: "outline" or "overlay". Determines the visualization style. Default is "outline".

`parent_image_suffix`
: Character string. Suffix expected for parent image files (e.g., "_Run.png"). Used to find parent images and derive base names. Default is "_Run.png".

mask_suffix	Character string. The final suffix part of the mask filename (e.g., "_cp_masks.png"). Default is "_cp_masks.png".
output_suffix	Character string. Suffix appended to the determined parent base name for the generated output files. Default is "_parent_overlayed.png". **Strongly recommend** not changing.
outline_col	Color for the outlines (used if mode="outline"). Default is "cyan".
outline_lwd	Approximate line width (thickness) for outlines in pixels (used if mode="outline"). Default is 2.
overlay_col	Color for the overlay (used if mode="overlay"). Default is "yellow".
overlay_alpha	Alpha transparency for the overlay (used if mode="overlay"). Default is 0.4.

Value

Invisibly returns a list of paths to the successfully created output files.

Examples

```
## Not run:
# --- Setup Dummy Files/Dirs ---
if (requireNamespace("magick", quietly = TRUE) && requireNamespace("tools", quietly = TRUE)) {
# Create temp directories
temp_dir <- tempdir()
parent_dir <- file.path(temp_dir, "parent_images")
mask_dir <- file.path(temp_dir, "crop_masks")
out_dir <- file.path(temp_dir, "parent_output")
dir.create(parent_dir, showWarnings = FALSE)
dir.create(mask_dir, showWarnings = FALSE)
dir.create(out_dir, showWarnings = FALSE)

# Define suffixes
parent_suffix <- "_Run.png" # Suffix for the parent image
mask_suffix <- "_cp_masks.png" # Suffix for the mask image
output_suffix <- "final_overlay.png"

# Create dummy parent image
parent_base_name <- "Experiment1_Tile1"
parent_file_path <- file.path(parent_dir, paste0(parent_base_name, parent_suffix))
parent_img <- magick::image_blank(width = 500, height = 400, color = "grey80")
magick::image_write(parent_img, parent_file_path)

# Create dummy mask file following the specific pattern
# cropped[parent_base_name]Run[mask_suffix]
mask_file_name_actual <- paste0("cropped", parent_base_name, "_Run", mask_suffix)
mask_path <- file.path(mask_dir, mask_file_name_actual)
mask <- magick::image_blank(width = 50, height = 50, color = "black")
mask <- magick::image_draw(mask)
graphics::symbols(25, 25, circles = 15, inches = FALSE, add = TRUE, fg = "white", bg = "white")
grDevices::dev.off()
magick::image_write(mask, mask_path)
```

```
# --- Example: coord_df with FULL filenames (handled by modified script) ---
coord_data_full <- data.frame(
  ImageFile = c(paste0(parent_base_name, parent_suffix)), # Full filename identifier
  X = c(300),
  Y = c(250)
)

overlayMaskOnParent(
  parent_image_input = parent_dir,
  mask_input = mask_dir,
  coord_df = coord_data_full, # Use the dataframe with full filenames
  output_target = out_dir,
  mode = "outline",
  parent_image_suffix = parent_suffix, # e.g., "_Run.png"
  mask_suffix = mask_suffix,         # e.g., "_cp_masks.png"
  output_suffix = output_suffix,
  outline_col = "blue"
)
print(paste("Output (full name input) saved in:", out_dir))
list.files(out_dir) # Should show one file named Experiment1_Tile1_final_overlay.png

# --- Clean up ---
unlink(parent_dir, recursive = TRUE)
unlink(mask_dir, recursive = TRUE)
unlink(out_dir, recursive = TRUE)
} else {
message("Magick and/or tools package not installed. Examples cannot run.")
}

## End(Not run)
```

Index