Package: segmantR (via r-universe)

November 3, 2026

Title Cell Segmentation for Histology with Human-in-the-Loop Training

Version 0.2.0

Description Cell segmentation for histology and multiplexed tissue imaging. Provides classical segmentation methods (adaptive thresholding, watershed, Voronoi propagation), wrappers through 'reticulate' for the deep learning models 'Cellpose' (Stringer et al. (2021) <doi:10.1038/s41592-020-01018-x>), 'StarDist' (Schmidt et al. (2018) <doi:10.1007/978-3-030-00934-2_30>) and 'Mesmer' (Greenwald et al. (2022) <doi:10.1038/s41587-021-01094-0>), and a human-in-the-loop iterative refinement workflow. A 'shiny' application lets researchers annotate, correct, train, evaluate and download custom segmentation models without leaving R. Versioned declarative segmentation protocols, a neutral interchange format (JSON manifests, integer label TIFF, GeoJSON, measurement tables with SHA-256 inventories), training dataset manifests with grouped splits, and readable 'QuPath' script templates with a 'StarDist' parameter adapter support reproducible exchange with 'QuPath' and partner packages.

License MIT + file LICENSE

URL https://github.com/CTTIR/segmantR, https://cttir.github.io/segmantR/

BugReports https://github.com/CTTIR/segmantR/issues

Encoding UTF-8

Language en-GB

SystemRequirements Optional: Python with the 'cellpose', 'stardist' or 'deepcell' modules for the deep learning backends; QuPath (>= 0.4) to run the exported QuPath scripts.

Roxygen list(markdown = TRUE)

Depends R (>= 4.1.0)

Imports digest, dplyr, rlang, sf (>= 1.0.0), stars (>= 0.6.0), cli, ggplot2 (>= 3.4.0), graphics, grDevices, jsonlite, shiny (>= 1.7.0), stats, tibble, tools, utils

Suggests callr, EBImage, htmlwidgets, imager, knitr, magick, nanoparquet, openssl, processx, reticulate (>= 1.35), rmarkdown, terra, testthat (>= 3.0.0), tiff, withr, zip

Config/testthat/edition 3

VignetteBuilder knitr

Config/Needs/website CTTIR/themakR

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev make libuv1-dev libssl-dev libproj-dev python3 libsqlite3-dev libudunits2-dev zlib1g-dev

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

Date/Publication 2026-10-29 11:19:01 UTC

RemoteUrl https://github.com/CTTIR/segmantR

RemoteRef HEAD

RemoteSha 882c2fb7fe0546f71f6f11bbfd3de6103859b098

Contents

new_sg_image . . . 5
new_sg_mask . . . 6
new_sg_trained_model . . . 8
sg_active_learning_loop . . . 9
sg_app . . . 10
sg_apply_corrections . . . 11
sg_canvas . . . 12
sg_canvas_correct . . . 13
sg_cleanup_labels . . . 14
sg_control_capabilities . . . 15
sg_create_annotation_task . . . 16
sg_dataset_manifest . . . 17
sg_display_levels . . . 18
sg_display_range . . . 20
sg_envi_info . . . 21
sg_evaluate_segmentation . . . 21
sg_example_image . . . 22
sg_example_mask . . . 23
sg_export_interchange . . . 24
sg_export_mask . . . 25
sg_export_qupath . . . 26
sg_extract_features . . . 27
sg_filter_cells . . . 28
sg_hash_assets . . . 29
sg_image_channels . . . 30
sg_import_interchange . . . 31
sg_import_qupath . . . 32

sg_interchange_capabilities . . . 33
sg_interchange_manifest . . . 34
sg_load_model . . . 35
sg_mask_legend . . . 36
sg_mask_review . . . 37
sg_mask_revision . . . 38
sg_mask_status . . . 39
sg_mask_to_polygons . . . 39
sg_merge_masks . . . 40
sg_model_card . . . 41
sg_package_model . . . 41
sg_plot_comparison . . . 43
sg_plot_features . . . 44
sg_plot_mask . . . 45
sg_plot_metrics . . . 45
sg_plot_overlay . . . 46
sg_predict_model . . . 47
sg_prepare_training_data . . . 48
sg_preprocess . . . 49
sg_protocol_get . . . 50
sg_protocol_list . . . 50
sg_protocol_run . . . 51
sg_protocol_schema . . . 52
sg_protocol_validate . . . 52
sg_read_envi . . . 53
sg_read_field . . . 54
sg_read_fields . . . 55
sg_read_image . . . 57
sg_read_window . . . 57
sg_roi . . . 58
sg_roi_add_layer . . . 59
sg_roi_add_roi . . . 59
sg_roi_area . . . 60
sg_roi_bbox . . . 61
sg_roi_buffer . . . 62
sg_roi_centroid . . . 62
sg_roi_check_containment . . . 63
sg_roi_check_geometry . . . 64
sg_roi_circle . . . 65
sg_roi_clamp . . . 66
sg_roi_contains . . . 67
sg_roi_dims . . . 67
sg_roi_distance . . . 68
sg_roi_ellipse . . . 69
sg_roi_fix_geometry . . . 70
sg_roi_flip_y . . . 71
sg_roi_frame . . . 71
sg_roi_freehand . . . 72

sg_roi_from_geojson . . . 73
sg_roi_from_sf . . . 74
sg_roi_layer . . . 75
sg_roi_layer_add . . . 76
sg_roi_layer_n . . . 77
sg_roi_layer_remove . . . 77
sg_roi_layer_rois . . . 78
sg_roi_layers . . . 79
sg_roi_mask . . . 79
sg_roi_mask_boundary . . . 81
sg_roi_mask_derive . . . 82
sg_roi_mask_legend . . . 83
sg_roi_mask_preview . . . 84
sg_roi_mask_stack . . . 84
sg_roi_mask_stats . . . 85
sg_roi_overlaps . . . 86
sg_roi_point . . . 87
sg_roi_polygon . . . 88
sg_roi_project . . . 89
sg_roi_read_geojson . . . 89
sg_roi_read_qupath . . . 90
sg_roi_rect . . . 92
sg_roi_remove_layer . . . 93
sg_roi_remove_roi . . . 93
sg_roi_rescale . . . 94
sg_roi_ring . . . 95
sg_roi_rois . . . 96
sg_roi_rois_overlap . . . 97
sg_roi_setops . . . 98
sg_roi_simplify . . . 99
sg_roi_snap . . . 100
sg_roi_style . . . 100
sg_roi_summary . . . 102
sg_roi_transform . . . 102
sg_roi_validate . . . 103
sg_roi_write_geojson . . . 104
sg_roi_write_qupath . . . 105
sg_run_app . . . 106
sg_segment_cellpose . . . 107
sg_segment_mesmer . . . 108
sg_segment_propagate . . . 109
sg_segment_stardist . . . 110
sg_segment_threshold . . . 111
sg_segment_watershed . . . 112
sg_select_channel . . . 114
sg_setup_python . . . 115
sg_slide_display_ranges . . . 116
sg_stain_deconvolve . . . 117

sg_stardist_capabilities 118
sg_stardist_manifest 119
sg_stardist_parameter_map 120
sg_train_cellpose 121
sg_train_stardist 122
sg_validate_interchange 123
sg_validate_training_manifest 124
sgCanvasOutput 125

Index **127**

`new_sg_image` *Create a new sg_image object*

Description

Constructor for the `sg_image` S3 class, which represents a multi-channel image with associated metadata.

Usage

```
new_sg_image(
  pixels,
  channels = NULL,
  resolution = NULL,
  metadata = list(),
  id = NULL,
  plane = NULL,
  origin = NULL,
  bands = NULL,
  value_semantics = "unknown",
  source_digest = NULL,
  transform_digest = NULL,
  provenance = list()
)
```

Arguments

`pixels`	Numeric array of dimensions H x W (grayscale) or H x W x C (multi-channel).
`channels`	Character vector of channel names. If `NULL`, defaults to `ch1`, `ch2`, etc.
`resolution`	Named list with `x_um` and `y_um`: microns per current array pixel. Divide these by `origin$downsample` for the full-resolution reference pixel sizes.
`metadata`	Named list of additional image metadata.
`id`	Optional stable image identifier (string). Absolute file paths must not be used as identity.
`plane`	Optional list with zero-based `level`, `series`, `c`, `z` and `t`. `c = NA` means that the array holds all channels.

origin	Optional list with x, y (position of the array's top-left corner in full-resolution pixel coordinates) and downsample (full-resolution pixels per array pixel).
bands	Optional data frame describing channels/bands with columns name, wavelength_nm, fwhm_nm (one row per channel).
value_semantics	What pixel values represent: one of "unknown", "intensity", "raw", "reflectance", "radiance", "absorbance", "probability".
source_digest	Optional "sha256:<hex>" digest of the source file or pixel content.
transform_digest	Optional digest of transformations applied to the source.
provenance	Named list of provenance records.

Details

The pixel array order is always [y, x, channel]: the first array row is the top image row. All arguments after metadata are optional interchange fields added in segmantR 0.2.0; objects created by older code simply lack them and are treated with the documented defaults.

Value

An object of class sg_image.

Examples

```
pixels <- matrix(runif(100), nrow = 10, ncol = 10)
img <- new_sg_image(pixels)
print(img)

# With interchange metadata
img2 <- new_sg_image(pixels, channels = "DAPI",
                     resolution = list(x_um = 0.5, y_um = 0.5),
                     id = "slide-001", value_semantics = "intensity")
```

---

new_sg_mask *Create a new sg_mask object*

Description

Constructor for the sg_mask S3 class, which stores an integer label matrix where 0 represents background and positive integers represent individual cell IDs.

Usage

```
new_sg_mask(
  labels,
  image_id = NULL,
  model_info = NULL,
  mask_type = c("instance", "labelled", "binary"),
  legend = NULL,
  plane = NULL,
  origin = NULL,
  status = c("draft", "staged", "reviewed"),
  provenance = list(),
  id = NULL
)
```

Arguments

labels	Integer matrix of cell labels. 0 = background, positive values = cell IDs (gaps are allowed). Values must be finite whole numbers from 0 to 2147483647; larger uint32 IDs are unsupported and rejected before integer conversion.
image_id	Optional character string identifying the source image.
model_info	Optional named list of model metadata.
mask_type	One of "instance", "labelled" or "binary".
legend	Optional data frame with one row per positive label and columns label, object_id, class, name. Missing legends are generated deterministically when needed (see sg_mask_legend()).
plane	Optional plane list (see new_sg_image()).
origin	Optional origin list (see new_sg_image()).
status	Review status: "draft" (default), "staged" or "reviewed". See sg_review_mask().
provenance	Named list of provenance records.
id	Optional mask identifier.

Details

Masks follow the segmantR mask contract: integer values, 0 is the background, positive integers are object instances (mask_type = "instance", the default and historic meaning), class codes ("labelled") or foreground ("binary", values 0/1 only). All arguments after model_info are optional interchange fields.

Value

An object of class sg_mask.

Examples

```
labels <- matrix(c(0L, 0L, 1L, 1L, 0L, 2L, 2L, 0L, 0L), nrow = 3)
mask <- new_sg_mask(labels)
print(mask)
```

`new_sg_trained_model` *Create a new sg_trained_model object*

Description

Constructor for the `sg_trained_model` S3 class, representing a trained segmentation model with associated metadata.

Usage

```
new_sg_trained_model(
  model_path,
  backend = c("cellpose", "stardist"),
  base_model,
  training_metrics,
  model_card = list(),
  runtime = NULL,
  dataset_manifest = NULL,
  license = NULL,
  representation = "unknown"
)
```

Arguments

`model_path`	Character. Path to the trained model weights.
`backend`	Character. Segmentation backend: `"cellpose"` or `"stardist"`.
`base_model`	Character. Name of the base model that was fine-tuned.
`training_metrics`	Named list of training metrics (e.g., loss curve, number of epochs).
`model_card`	Named list of model card metadata (description, author, intended use, etc.). The versioned card written by `sg_package_model()` reads `purpose`, `data_domain`, `channels`, `target_pixel_size_um`, `input_value_semantics`, `normalization`, `limitations` (character), `evaluation` (named list of metrics) and `license` from here.
`runtime`	Optional runtime description (list) recorded at training time; never a copy of a Python environment.
`dataset_manifest`	Optional `sg_dataset_manifest` the model was trained on.
`license`	Optional list with `id` (e.g. an SPDX identifier), `name`, `source` and `text`.
`representation`	Weight representation: `"pb"`, `"savedmodel"`, `"bioimageio"`, `"pytorch"`, `"keras"`, `"directory"`, `"file"` or `"unknown"`.

Value

An object of class `sg_trained_model`.

Examples

```
# a dedicated directory with a stand-in weight file
weights <- file.path(tempfile("model"), "cellpose_weights")
dir.create(dirname(weights))
writeBin(as.raw(1:16), weights)
mdl <- new_sg_trained_model(
  model_path = weights,
  backend = "cellpose",
  base_model = "cyto3",
  training_metrics = list(n_epochs = 100L, final_loss = 0.05)
)
print(mdl)
```

---

sg_active_learning_loop

Run an active learning loop for iterative segmentation refinement

---

Description

Orchestrates a human-in-the-loop (HITL) active learning workflow: segment, review, correct, retrain, repeat.

Usage

```
sg_active_learning_loop(
  image,
  model = "cellpose",
  n_rounds = 5L,
  patches_per_round = 10L,
  patch_size = 256L,
  initial_model = "cyto3"
)
```

Arguments

image	An sg_image object.
model	Character. Segmentation backend to use. Currently only "cellpose" is supported. Default "cellpose".
n_rounds	Integer. Number of active learning rounds. Default 5L.
patches_per_round	Integer. Patches sampled per round. Default 10L.
patch_size	Integer. Patch size in pixels. Default 256L.
initial_model	Character. Initial model name. Default "cyto3".

Value

An `sg_hitl_result` S3 object with elements `$rounds` (list of per-round results), `$final_model` (trained model path or object), and `$metrics` (training metrics across rounds).

Examples

```
## Not run:
# Requires Python with Cellpose and an interactive review session
img <- sg_example_image("fluorescence_nuclei")
result <- sg_active_learning_loop(img, n_rounds = 3L,
                                  patches_per_round = 2L, patch_size = 32L)

## End(Not run)
```

---

sg_app *Build the segmantR Shiny application*

Description

Returns a `shiny.appobj` built from package code, with explicit input objects instead of global options. The app offers image/channel/band selection, a protocol picker with a parameter form generated from the protocol registry, mask overlay, backend status for optional DNN protocols, a staged/reviewed workflow with explicit overwrite confirmation, and export of a neutral interchange bundle. There is no code entry of any kind.

Usage

```
sg_app(input = NULL, state = NULL, control = "off", ...)
```

Arguments

`input`	`NULL`, an `sg_image`, an `sg_mask`, a list with `image` and/or `mask`, or the path of an interchange bundle (imported as staged).
`state`	Optional list with initial `protocol` (id) and `parameters` (named list of overrides).
`control`	`"off"`. The local control service (`segmantR-control-v1`) is planned and not implemented; other values signal an `sg_capability_error`.
`...`	Options: `read_only` (logical, disables running and review).

Value

A `shiny.appobj`.

See Also

`sg_run_app()`, `sg_control_capabilities()`

Examples

```
app <- sg_app(list(image = sg_example_image("fluorescence_nuclei")))
class(app)
if (interactive()) {
  shiny::runApp(app)
}
```

`sg_apply_corrections` *Apply manual corrections to a segmentation mask*

Description

Takes a list of correction operations (split, merge, delete, add) and applies them to produce a corrected mask.

Usage

```
sg_apply_corrections(mask, corrections)
```

Arguments

`mask`	An `sg_mask` object.
`corrections`	A list of correction operations. Each element is a list with at least an `$action` field (`"split"`, `"merge"`, `"delete"`, or `"add"`). See Details.

Details

Each correction in the list must contain:

`action = "delete"` Remove cell. Requires `$cell_id`.

`action = "merge"` Merge cells. Requires `$cell_ids` (integer vector).

`action = "split"` Split a cell. Requires `$cell_id` and `$seed_points` (matrix with columns row, col).

`action = "add"` Add a new cell. Requires `$pixels` (matrix of row/col coordinates).

Value

A corrected `sg_mask` object.

Examples

```
labels <- matrix(0L, nrow = 20, ncol = 20)
labels[3:8, 3:8] <- 1L
labels[12:18, 12:18] <- 2L
mask <- new_sg_mask(labels)
corrected <- sg_apply_corrections(mask, list(
  list(action = "delete", cell_id = 2L)
))
print(corrected)
```

sg_canvas *Draw cell boundaries on a microscopy image*

Description

The canvas uses local array pixel coordinates (top-left origin, x right, y down). Drawing emits Shiny inputs suffixed `_created`, `_edited`, and `_erased`. Use `sg_canvas_correct()` to stage changes to a mask.

Usage

```
sg_canvas(
  image,
  mask = NULL,
  channel = 1L,
  tool = "pan",
  width = NULL,
  height = 600,
  elementId = NULL
)
```

Arguments

`image`	An `sg_image`.
`mask`	Optional `sg_mask` bound to the same image.
`channel`	One-based display channel.
`tool`	Drawing tool.
`width, height`	Widget dimensions.
`elementId`	Optional widget id.

Value

An HTML widget.

Examples

```
img <- new_sg_image(matrix(seq_len(72), 8, 9), id = "slide")
msk <- new_sg_mask(matrix(0L, 8, 9), image_id = "slide")
widget <- sg_canvas(img, msk, tool = "polygon")
class(widget)
if (interactive()) widget
```

`sg_canvas_correct` *Apply a cell canvas correction*

Description

Geometry is in local array pixels. Changes retain image binding and are staged for explicit review. Overlapping cells are rejected.

Usage

```
sg_canvas_correct(
  mask,
  geometry = NULL,
  action = "add",
  cell_id = NULL,
  expected_revision
)
```

Arguments

`mask`	An `sg_mask`.
`geometry`	GeoJSON Polygon or MultiPolygon, required for add/edit.
`action`	One of `add`, `edit`, `erase`.
`cell_id`	Existing integer label for edit/erase.
`expected_revision`	Revision shown when the edit began.

Value

A staged `sg_mask`.

Examples

```
msk <- new_sg_mask(matrix(0L, 8, 9))
box <- list(type = "Polygon",
            coordinates = list(list(c(1, 1), c(4, 1), c(4, 4), c(1, 4), c(1, 1))))
added <- sg_canvas_correct(msk, box, expected_revision = sg_mask_revision(msk))
sum(added$labels > 0)
added$review$status
```

`sg_cleanup_labels` *Clean up a label mask*

Description

Applies the `postprocess.label-cleanup.v1` rules to an `sg_mask` in a fixed order: handling of disconnected label parts, hole filling, separation of touching objects, border removal, area filtering and relabelling. The legend (object ids, classes) follows the labels.

Usage

```
sg_cleanup_labels(
  mask,
  min_area = 0L,
  max_area = NULL,
  min_area_um2 = NULL,
  max_area_um2 = NULL,
  fill_holes = FALSE,
  connectivity = 4L,
  border = c("keep", "remove"),
  touching = c("keep", "separate"),
  disconnected = c("keep", "split", "keep_largest"),
  relabel = c("value", "raster", "none"),
  pixel_size = NULL
)
```

Arguments

`mask`	An `sg_mask` object.
`min_area`	Minimum object area in pixels (full-resolution pixels for downsampled masks are not assumed; areas are array pixels).
`max_area`	Maximum object area in pixels, or `NULL` for no limit.
`min_area_um2, max_area_um2`	Area limits in square micrometres; require `pixel_size`.
`fill_holes`	Logical; fill background holes enclosed by one label.
`connectivity`	`4L` or `8L`; used for hole detection and disconnected-part handling.
`border`	`"keep"` or `"remove"` objects touching the array border.
`touching`	`"keep"` or `"separate"`; `"separate"` sets contact pixels of the higher label to background (4-neighbourhood).
`disconnected`	`"keep"`, `"split"` (new labels for extra parts) or `"keep_largest"` for labels consisting of several parts. Splitting fails if the additional labels would exceed the supported signed-integer range.
`relabel`	`"value"` (1..N keeping order), `"raster"` (1..N by first appearance) or `"none"`.
`pixel_size`	Optional list with `x` and `y` in micrometres.

Value

A new sg_mask. Masks derived from staged or reviewed masks are staged.

Examples

```
labels <- matrix(0L, 12, 12)
labels[2:4, 2:4] <- 1L
labels[7:11, 7:11] <- 2L
labels[9, 9] <- 0L
out <- sg_cleanup_labels(new_sg_mask(labels), min_area = 10L,
                         fill_holes = TRUE)
out$n_cells
```

---

sg_control_capabilities

Control service capabilities (segmantR-control-v1)

Description

The local control service lets a partner (qupflowR, hyperannotatR) drive a running segmantR app through typed commands. It is **planned** and not implemented; this function reports the planned contract so partners can check for it without guessing. It never starts a server.

Usage

```
sg_control_capabilities()
```

Value

A list with schema, status ("planned"), implemented (FALSE), default ("off"), planned_commands and security_requirements.

Examples

```
sg_control_capabilities()$status
```

`sg_create_annotation_task` *Create an annotation task for human review*

Description

Samples informative image patches for manual annotation or correction. Patches are selected on a grid or, when an initial mask is provided, prioritised by segmentation uncertainty.

Usage

```
sg_create_annotation_task(
  image,
  initial_mask = NULL,
  region = NULL,
  n_patches = 20L,
  patch_size = 256L
)
```

Arguments

`image`	An `sg_image` object.
`initial_mask`	Optional `sg_mask` object. When provided, patches are scored by segmentation uncertainty and sampled accordingly.
`region`	Optional numeric vector `c(row_min, row_max, col_min, col_max)` to restrict patch sampling to a sub-region. Patch coordinates are then relative to the region, and `initial_mask` is cropped to it.
`n_patches`	Integer. Number of patches to sample. Default `20L`.
`patch_size`	Integer. Size of each square patch in pixels. Default `256L`.

Value

An `sg_annotation_task` S3 object (a list with elements `patches`, `image_id`, `n_patches`, `patch_size`, and `created`).

Examples

```
pixels <- matrix(runif(100 * 100), nrow = 100, ncol = 100)
img <- new_sg_image(pixels)
task <- sg_create_annotation_task(img, n_patches = 4L, patch_size = 32L)
print(task)
```

`sg_dataset_manifest` *Describe a training dataset*

Description

Builds a tile-level dataset manifest from image/mask pairs without writing pixels. Tiles are laid out on a regular grid (`tile_size`, `overlap`), splits are assigned per **group** (`subject_id`, `sample_id` or `slide_id`) with a fixed seed so tiles of one origin never land in different splits, and every tile records its bounds, plane, object list (label, object id, class) and exclusion reason.

Usage

```
sg_dataset_manifest(
  pairs,
  tile_size = 256L,
  overlap = 0L,
  group_by = c("subject_id", "sample_id", "slide_id"),
  split = c(train = 0.7, validation = 0.15, test = 0.15),
  seed = 1L,
  mask_type = c("instance", "labelled", "binary"),
  min_objects = 1L,
  partial_tiles = c("exclude", "keep"),
  require_reviewed = FALSE,
  normalization = list(method = "none"),
  backend = NULL
)
```

Arguments

`pairs`	A list of training pairs. Each element is a list with `image` (`sg_image`), `mask` (`sg_mask`) and the grouping ids `subject_id`, `sample_id` and/or `slide_id`.
`tile_size`	Tile edge length in pixels.
`overlap`	Overlap between neighbouring tiles in pixels.
`group_by`	Grouping key used for the split.
`split`	Named proportions for `train`, `validation` and `test` (normalised to sum 1).
`seed`	Integer seed of the group permutation (the global RNG state is restored afterwards).
`mask_type`	Required mask type.
`min_objects`	Tiles with fewer objects are excluded (`"empty"` for zero objects, `"too_few_objects"` otherwise).
`partial_tiles`	`"exclude"` border tiles smaller than `tile_size`, or `"keep"` them.
`require_reviewed`	Logical; exclude tiles from masks that are not reviewed (`"not_reviewed"`).
`normalization`	List with `method` (`"none"`, `"minmax"`, `"percentile"`), `low`, `high` describing the normalisation the trainer applies (recorded, not applied here).
`backend`	Optional target backend name recorded in the manifest.

Details

Masks must satisfy the requested `mask_type`: integer, background 0, and for `"instance"` one connected part per label. A foreground/background mask with several objects under one label is rejected (`NOT_INSTANCE_MASK`) instead of being accepted silently.

Value

An `sg_dataset_manifest` list conforming to `dataset-manifest.schema.json`.

Examples

```
img <- sg_example_image("fluorescence_nuclei")
# The synthetic example mask contains overlapping ellipses; split them
# into separate instances first.
msk <- sg_cleanup_labels(sg_example_mask("fluorescence_nuclei"),
                         disconnected = "split")
pairs <- list(
  list(image = img, mask = msk, subject_id = "A"),
  list(image = img, mask = msk, subject_id = "B")
)
man <- sg_dataset_manifest(pairs, tile_size = 32L, min_objects = 0L,
                           split = c(train = 0.5, validation = 0.5,
                                     test = 0))
table(vapply(man$tiles, function(t) t$split, ""))
```

---

sg_display_levels *Display levels from positive and negative reference cells*

Description

Sets the black and full-colour levels of each channel from the signal around reference cells, so that called-negative cells stay dark and called-positive cells reach full colour: for every channel up to `n` negative and `n` positive cells are drawn, a statistic of the `(2 * half + 1)`-pixel square patch around each cell is taken (`"mean"` or a percentile given as a probability), and the levels are the `black` quantile of the negative and the `white` quantile of the positive cell values.

Usage

```
sg_display_levels(
  path,
  channels,
  cells,
  positive,
  negative,
  x = "x",
  y = "y",
  label = "label",
  pixel_size = 1,
```

```
  half = 7,
  statistic = 0.9,
  black = 0.5,
  white = 0.5,
  n = 300,
  seed = 0
)
```

Arguments

path	Path to the image file.
channels	Channel names.
cells	Data frame of cells with positions and labels.
positive, negative	Named lists: channel name to the labels of the positive and negative reference cells.
x, y, label	Column names in cells (positions in physical units).
pixel_size	Physical size of one pixel.
half	Patch half-width in pixels.
statistic	"mean" or a probability for a patch percentile.
black, white	Quantiles across negative and positive cells.
n	Maximum number of reference cells per set.
seed	Random seed for the draws (the global RNG state is restored).

Details

Arguments that vary by channel (half, statistic, black, white) accept a single value or a named list/vector by channel name, with an optional .default entry.

Value

A data frame with columns channel, lo, hi, n_negative and n_positive.

See Also

sg_slide_display_ranges(), sg_display_range()

Examples

```
f <- tempfile(fileext = ".tif")
plane <- matrix(10, 40, 40)
plane[21:40, ] <- 200
invisible(tiff::writeTIFF(plane / 255, f, bits.per.sample = 8L))
cells <- data.frame(x = c(5, 15, 25, 35), y = c(5, 10, 30, 35),
                    label = c("neg", "neg", "pos", "pos"))
sg_display_levels(f, "ch1", cells, positive = list(ch1 = "pos"),
                  negative = list(ch1 = "neg"), half = 2)
```

`sg_display_range` *Display range of fluorescence values*

Description

Percentile-based display (black and white) levels for a set of pixel values, as used to scale fluorescence channels for viewing: `lo` is the lower percentile (or a given value), `hi` the larger of the upper percentile, an optional floor and `lo` + `min_span`. Percentiles use linear interpolation between order statistics (`stats::quantile(type = 7)`).

Usage

```
sg_display_range(
  x,
  probs = c(0.01, 0.997),
  lo = NULL,
  hi_floor = NULL,
  min_span = 1
)
```

Arguments

`x`	Numeric pixel values (any shape).
`probs`	Lower and upper probabilities; either may be `NA` when `lo` or `hi_floor` replace it.
`lo`	Optional fixed lower level (replaces the lower percentile).
`hi_floor`	Optional value the upper level must at least reach.
`min_span`	Minimum distance between the two levels.

Value

Named numeric `c(lo = , hi = )`.

See Also

`sg_slide_display_ranges()`, `sg_display_levels()`

Examples

```
sg_display_range(c(0, 1, 2, 3, 100))
sg_display_range(1:10, probs = c(NA, 0.995), lo = 2, hi_floor = 20)
```

sg_envi_info *Read and validate an ENVI header without reading pixel data*

Description

Read and validate an ENVI header without reading pixel data

Usage

```
sg_envi_info(path, data_file = NULL)
```

Arguments

path	Path to the .hdr file or to the data file next to it.
data_file	Optional explicit data file path.

Value

A list of class sg_envi_info with samples, lines, bands, interleave, byte_order, data_type, dtype, header_offset, wavelength_nm, fwhm_nm, band_names, nodata, scale_factor, gain, offset, wavelength_units, data_bytes_expected, data_bytes_present and header_digest. Paths are not stored.

Examples

```
d <- tempfile("envi")
dir.create(d)
hdr <- file.path(d, "cube.hdr")
writeLines(c("ENVI", "samples = 2", "lines = 2", "bands = 2",
             "header offset = 0", "data type = 12", "interleave = bsq",
             "byte order = 0", "wavelength = {500, 600}"), hdr)
writeBin(1:8, file.path(d, "cube"), size = 2L, endian = "little")
sg_envi_info(hdr)$wavelength_nm
```

sg_evaluate_segmentation
Evaluate segmentation quality

Description

Computes standard instance segmentation metrics by comparing a predicted mask against a ground truth mask.

Usage

```
sg_evaluate_segmentation(
  predicted,
  ground_truth,
  metrics = c("dice", "jaccard", "aji", "panoptic_quality", "ap50", "ap75",
    "f1_detection")
)
```

Arguments

predicted	An sg_mask object with predicted segmentation.
ground_truth	An sg_mask object with ground truth segmentation.
metrics	Character vector of metrics to compute. One or more of "dice", "jaccard", "aji", "panoptic_quality", "ap50", "ap75", "f1_detection". Default is all.

Details

Instances are matched one-to-one at an intersection-over-union (IoU) threshold. "ap50" and "ap75" are the instance-segmentation average precision used by Cellpose and StarDist at IoU 0.5 and 0.75, $TP/(TP+FP+FN)$; "f1_detection" is the F1 score of the matches at IoU 0.5, $2TP/(2TP+FP+FN)$.

Value

An sg_eval S3 object (a tibble with columns metric and value) with a custom print method.

Examples

```
pred_labels <- matrix(0L, nrow = 20, ncol = 20)
pred_labels[3:8, 3:8] <- 1L
pred_labels[12:18, 12:18] <- 2L
gt_labels <- matrix(0L, nrow = 20, ncol = 20)
gt_labels[3:9, 3:9] <- 1L
gt_labels[12:17, 12:17] <- 2L
pred <- new_sg_mask(pred_labels)
gt <- new_sg_mask(gt_labels)
result <- sg_evaluate_segmentation(pred, gt)
print(result)
```

---

sg_example_image *Load a bundled example image*

Description

Returns an sg_image containing synthetic image data for testing and demonstrations. The data are drawn with a fixed seed, so they are the same in every session, and the caller's random number stream is left unchanged.

Usage

```
sg_example_image(type = c("he_breast", "fluorescence_nuclei", "multiplex_4ch"))
```

Arguments

type	Character, the example to generate: • "he_breast": 64 x 64 single-channel image simulating an H&E stained breast section with about 20 cell-like blobs; • "fluorescence_nuclei": 64 x 64 single-channel image simulating DAPI-stained nuclei, about 15 bright spots on a dark background; • "multiplex_4ch": 64 x 64 x 4 array simulating a four-channel panel (DAPI, CD3, CD8, PanCK) with about 18 cells.

Value

An sg_image object.

See Also

sg_example_mask() for the matching label masks.

Examples

```
img <- sg_example_image("he_breast")
print(img)
```

---

sg_example_mask *Load a bundled example mask*

Description

Returns an sg_mask matching the corresponding example image. Like sg_example_image(), it leaves the caller's random number stream unchanged.

Usage

```
sg_example_mask(type = c("he_breast", "fluorescence_nuclei", "multiplex_4ch"))
```

Arguments

type	Character, one of "he_breast", "fluorescence_nuclei", or "multiplex_4ch" (see sg_example_image()).

Value

An sg_mask object whose labels are 0 for background and positive integers for the cells of the example image of the same type.

Examples

```
mask <- sg_example_mask("he_breast")
print(mask)
```

`sg_export_interchange` *Export to the neutral interchange format*

Description

Writes a bundle directory containing a manifest, the requested representations and an `integrity.json` inventory with SHA-256 hashes and relative paths. All representations describe the same envelope: the integer mask TIFF holds exact labels, the GeoJSON holds exact pixel-edge polygons in image coordinates with the same object ids, and the measurements table uses the canonical long format (`value_state` distinguishes missing, NaN and infinite values). RDS is an optional R-native optimisation and never the only representation.

Usage

```
sg_export_interchange(
  x,
  destination,
  formats = c("manifest", "mask_tiff", "geojson", "measurements"),
  ...,
  image = NULL,
  object_type = c("detection", "annotation", "cell"),
  overwrite = FALSE
)
```

Arguments

`x`	An `sg_mask`, `sg_run`, `sg_image`, or a named list with `image`, `mask`, `run`, `measurements`.
`destination`	Directory to create. It must not exist or be empty; with `overwrite = TRUE` an existing segmantR bundle is replaced.
`formats`	Any of `"manifest"`, `"mask_tiff"`, `"geojson"`, `"measurements"`, `"measurements_parquet"`, `"legend"`, `"image_tiff"`, `"run"`, `"rds"`. The manifest and integrity inventory are always written.
`...`	Reserved; must be empty.
`image`	Optional `sg_image` for image descriptors, intensity measurements and `"image_tiff"`.
`object_type`	QuPath object type for GeoJSON features (`"detection"`, `"annotation"` or `"cell"`).
`overwrite`	Logical; replace an existing segmantR bundle.

Value

An `sg_interchange_bundle` (invisibly) with `path`, `manifest` and `bundle_digest`.

Examples

```
m <- sg_example_mask("fluorescence_nuclei")
img <- sg_example_image("fluorescence_nuclei")
out <- sg_export_interchange(m, tempfile("bundle"), image = img)
list.files(out$path)
out$bundle_digest
```

`sg_export_mask` *Export mask to file*

Description

Writes a segmentation mask to disk in one of several formats.

Usage

```
sg_export_mask(
  mask,
  path,
  format = c("tiff", "png", "geojson", "qupath", "csv")
)
```

Arguments

`mask`	An `sg_mask` object.
`path`	Character. Output file path.
`format`	Character. Export format: `"tiff"` (default), `"png"`, `"geojson"`, `"qupath"`, or `"csv"`.

Value

The output file path, returned invisibly.

Examples

```
labels <- matrix(0L, nrow = 10, ncol = 10)
labels[3:8, 3:8] <- 1L
mask <- new_sg_mask(labels)
tmp <- tempfile(fileext = ".csv")
sg_export_mask(mask, tmp, format = "csv")
```

sg_export_qupath *Export segmantR objects or a StarDist protocol for QuPath*

Description

Writes a QuPath program directory: a declarative `run.json` (schema `segmantR-qupath-run-v1`), the readable Groovy templates, the resources they bind (a neutral interchange bundle or a hashed model copy), a `run_export.json` for the way back, `checksums.sha256` and a `README.md` with the interactive and command-line invocations. segmantR never starts QuPath, Java or Groovy itself.

Usage

```
sg_export_qupath(
  x,
  destination,
  ...,
  image = NULL,
  image_name = NULL,
  model = NULL,
  region_policy = c("whole_image", "selected_annotations", "all_annotations"),
  save_policy = c("none", "project"),
  object_type = c("detection", "annotation", "cell"),
  require_calibration = TRUE,
  min_qupath_version = "0.5.0",
  allow_unsupported = FALSE,
  overwrite = FALSE
)
```

Arguments

`x`	An `sg_mask`, `sg_run`, list with `mask`/`image`, or a StarDist protocol id / `sg_protocol`.
`destination`	Output directory (must not exist or be empty).
`...`	Protocol parameter overrides for the `stardist` task.
`image`	`sg_image` describing the QuPath image (size, pixel size, plane, channels). Required for the `stardist` task.
`image_name`	Name of the image in the QuPath project (checked by the templates); `NULL` skips the name check.
`model`	StarDist model file/directory or `sg_trained_model` for the `stardist` task.
`region_policy`	`"whole_image"`, `"selected_annotations"` or `"all_annotations"`.
`save_policy`	`"none"` (default) or `"project"`.
`object_type`	QuPath object type for imported or exported objects.
`require_calibration`	Logical; templates refuse uncalibrated images.

`min_qupath_version`
: Minimum QuPath version written to `run.json`.

`allow_unsupported`
: Logical; allow non-default parameters that the QuPath extension cannot honour (they are still listed).

`overwrite`
: Logical; replace an existing program directory created by segmantR.

Details

- For an `sg_mask`, `sg_run` or list with a mask, the task is `import`: the mask is exported with `sg_export_interchange()` into `bundle/` (integer TIFF, exact GeoJSON with object ids, legend, measurements) and `segmantR_import.groovy` adds the objects to the open image after checking size, calibration, plane, SHA-256 and id collisions.
- For a StarDist protocol (`"stardist.2d.v1"` or an `sg_protocol`), the task is `stardist`: parameters are mapped with `sg_stardist_parameter_map()` (unsupported non-default options are errors unless `allow_unsupported = TRUE`, and are always listed in `run.json`), the model is copied into `models/` and its digest is recorded for verification by the template.

Value

An `sg_qupath_program` (invisibly) with `path`, `task`, `run` (parsed `run.json`) and `commands` (headless command lines).

Examples

```
m <- sg_example_mask("fluorescence_nuclei")
img <- sg_example_image("fluorescence_nuclei")
prog <- sg_export_qupath(m, tempfile("qupath"), image = img,
                         require_calibration = FALSE)
prog$commands[["import"]]
```

---

sg_extract_features *Extract per-cell features from an image and mask*

Description

Computes intensity, morphology, texture, and location features for each segmented cell. Returns a tibble with one row per cell.

Usage

```
sg_extract_features(
  image,
  mask,
  features = c("intensity", "morphology", "texture", "location"),
  channels = NULL
)
```

Arguments

image	An sg_image object.
mask	An sg_mask object with labels matching the image dimensions.
features	Character vector of feature groups to compute. One or more of "intensity", "morphology", "texture", "location". Default is all four.
channels	Integer or character vector selecting which image channels to use for intensity and texture features. NULL (default) uses all channels.

Details

Morphology features: perimeter is the number of boundary pixels; circularity is $4\pi A/P^2$, capped at 1 because the boundary-pixel count underestimates the perimeter of round objects; eccentricity comes from the eigenvalues of the pixel-coordinate covariance (0 for a disc, 1 for a line of pixels); solidity is the area divided by the area of the convex hull of the pixel squares. Circularity, eccentricity and solidity are NA for single-pixel cells.

Value

A tibble with one row per cell and a cell_id column, plus columns for each requested feature.

Examples

```
pixels <- array(runif(20 * 20 * 2), dim = c(20, 20, 2))
img <- new_sg_image(pixels, channels = c("DAPI", "CD3"))
labels <- matrix(0L, nrow = 20, ncol = 20)
labels[3:8, 3:8] <- 1L
labels[12:18, 12:18] <- 2L
mask <- new_sg_mask(labels)
feats <- sg_extract_features(img, mask, features = c("intensity", "morphology"))
print(feats)
```

---

sg_filter_cells *Filter cells by morphological criteria*

Description

Removes cells from a segmentation mask that fall outside specified morphological thresholds. Useful for quality control after initial segmentation.

Usage

```
sg_filter_cells(
  mask,
  min_area = 50L,
  max_area = 5000L,
  min_circularity = 0.3,
  max_eccentricity = 0.95,
  border_cells = c("keep", "remove", "flag")
)
```

Arguments

mask	An sg_mask object.
min_area	Integer. Minimum cell area in pixels. Default 50L.
max_area	Integer. Maximum cell area in pixels. Default 5000L.
min_circularity	Numeric in [0, 1]. Minimum circularity (4 * pi * area / perimeter^2). Default 0.3.
max_eccentricity	Numeric in [0, 1]. Maximum eccentricity. Default 0.95.
border_cells	Character. How to handle cells touching the image border: "keep" (default), "remove", or "flag".

Value

A filtered sg_mask object. When border_cells = "flag", the returned mask has an additional border_cell_ids element.

Examples

```
labels <- matrix(0L, nrow = 20, ncol = 20)
labels[3:8, 3:8] <- 1L
labels[12:18, 12:18] <- 2L
mask <- new_sg_mask(labels)
filtered <- sg_filter_cells(mask, min_area = 10L, max_area = 100L)
print(filtered)
```

---

sg_hash_assets *Hash, write or verify a bundle inventory*

Description

Computes SHA-256 digests and sizes for every regular file below root (except the inventory files themselves), using bundle-relative forward-slash paths. The integrity.json form is identical to the qupflowR workflow-bundle rule: {"format_version": "1.0", "files": [{"path", "size_bytes", "sha256"}]} sorted by path, and the bundle digest is the SHA-256 of its canonical JSON (keys sorted, no whitespace). checksums.sha256 uses the sha256sum text format.

Usage

```
sg_hash_assets(
  root,
  files = NULL,
  write = c("none", "integrity", "sha256sum", "both"),
  verify = FALSE
)
```

Arguments

root	Bundle directory.
files	Optional character vector of relative paths to inventory (default: all regular files).
write	Which inventory files to write: "none", "integrity", "sha256sum" or "both".
verify	Logical; if TRUE, compare the directory with its existing integrity.json (or checksums.sha256) and signal an sg_integrity_error on any missing, extra, duplicate, resized or modified file.

Details

Symbolic links, absolute or parent-relative paths and special files are rejected. Absolute paths are never written.

Value

An sg_asset_inventory list with files (tibble), integrity (list) and bundle_digest ("sha256:<hex>"), invisibly when writing.

Examples

```
d <- tempfile("bundle")
dir.create(d)
writeLines("a", file.path(d, "a.txt"))
inv <- sg_hash_assets(d, write = "integrity")
inv$bundle_digest
sg_hash_assets(d, verify = TRUE)$bundle_digest == inv$bundle_digest
```

---

sg_image_channels *List the channels of a multi-page microscopy TIFF*

Description

Reads the directory structure of a (Q)PTIFF, OME-TIFF or plain multi-page TIFF through GDAL and returns one row per full-resolution channel plane. Channel names come from the per-plane image description (<Biomarker>, then <Name>, as written by PhenoCycler/Vectra QPTIFF files), from the Channel Name attributes of OME-XML on the first plane, or default to ch1, ch2, ... Reduced-resolution pages (overviews, thumbnails, label and macro images) are left out: a plane belongs to the channel set when it has one band and the size of the first plane.

Usage

```
sg_image_channels(path)
```

Arguments

path	Path to the image file.

Value

A data frame with columns `channel` (1-based channel number), `name`, `directory` (1-based TIFF directory), `width`, `height`, `block_width`, `block_height`, `datatype` and `dsn` (the GDAL dataset name of the plane).

See Also

`sg_read_window()`, `sg_read_field()`

Examples

```
f <- tempfile(fileext = ".tif")
planes <- list(matrix(1:100, 10, 10), matrix(101:200, 10, 10))
invisible(tiff::writeTIFF(lapply(planes, function(p) p / 255), f,
                          bits.per.sample = 8L))
sg_image_channels(f)[, c("channel", "name", "width", "height")]
```

---

sg_import_interchange *Import interchange data*

Description

Reads a segmantR interchange bundle or a single GeoJSON, integer TIFF, measurement CSV or RDS file, verifies it and returns a report with the imported objects. Imported masks are always **staged**; they never replace reviewed data (see `sg_replace_mask()`).

Usage

```
sg_import_interchange(
  path,
  format = "auto",
  expected_digest = NULL,
  ...,
  shape_yx = NULL,
  image = NULL,
  origin = NULL,
  trust_rds = FALSE,
  error = TRUE
)
```

Arguments

`path`	A bundle directory, its `manifest.json`, or a `.geojson`, `.json`, `.tif`/`.tiff`, `.csv` or `.rds` file.
`format`	`"auto"`, `"bundle"`, `"geojson"`, `"mask_tiff"`, `"measurements"` or `"rds"`.

expected_digest	Optional bundle digest ("sha256:<hex>") the inventory must match.
...	Reserved; must be empty.
shape_yx	Image shape needed to rasterise a standalone GeoJSON.
image	Optional sg_image providing shape, origin, plane and id for standalone files.
origin	Optional origin list for standalone GeoJSON.
trust_rds	Logical; RDS files are only deserialised when TRUE (after integrity checks for bundles).
error	Logical; if FALSE, failed checks are reported instead of signalled.

Details

Bundle checks: inventory (missing, extra, duplicate, resized or modified files, unsafe paths, symbolic links), optional expected_digest, schema family and major version, JSON Schema, asset hashes, TIFF dtype, shape and orientation, background code, label legend completeness, plane consistency, content digest and revision, GeoJSON object ids and exact geometry agreement with the TIFF, and measurement object ids.

Value

An sg_import_report list with ok, format, checks (tibble), mask (staged sg_mask or NULL), image, measurements, features (parsed GeoJSON features), manifest, bundle_digest, conversions and error.

Examples

```
m <- sg_example_mask("fluorescence_nuclei")
b <- sg_export_interchange(m, tempfile("bundle"))
rep <- sg_import_interchange(b$path, expected_digest = b$bundle_digest)
rep$ok
sg_mask_status(rep$mask)$status
```

---

sg_import_qupath *Import QuPath results into segmantR*

Description

Reads the output of the segmantR QuPath templates (a directory with a qupath-export manifest and integrity.json), a QuPath program directory containing such an output (out/ or qupath_export/), or a plain QuPath GeoJSON export. All checks of sg_import_interchange() run (inventory, schema, dtype, shape/orientation, legend, content digest and revision); in addition the QuPath GeoJSON is rasterised and compared with the QuPath label TIFF, object ids are matched through the legend, the run id and image binding can be checked, and measurements are imported in the canonical long format (namespace = "stored"). The mask is staged.

Usage

```
sg_import_qupath(path, ..., image = NULL, expected_run_id = NULL, error = TRUE)
```

Arguments

path	Export directory, program directory or GeoJSON file.
...	Reserved; must be empty.
image	Optional sg_image to check the binding (size, pixel size, plane) or to rasterise a plain GeoJSON.
expected_run_id	Optional run id (run.json run_id) the export must belong to.
error	Logical; signal failures instead of reporting them.

Value

An sg_import_report with an extra qupath element (QuPath and extension versions, template version, run id, GeoJSON/TIFF agreement).

Examples

```
# a plain QuPath GeoJSON export, rasterised onto the image grid
cells <- sg_roi_layer_add(sg_roi_layer("cells"),
                          sg_roi_rect(2, 2, 8, 8, label = "cell"))
cells <- sg_roi_layer_add(cells, sg_roi_rect(10, 10, 15, 16, label = "cell"))
p <- tempfile(fileext = ".geojson")
sg_roi_write_qupath(sg_roi_project(sg_roi_frame(c(20, 20)), cells), p)
rep <- sg_import_qupath(p, image = new_sg_image(matrix(0, 20, 20)))
rep$mask$n_cells
```

---

sg_interchange_capabilities

Interchange capability report

Description

Describes what this installation of segmantR can exchange: schema and API versions, interoperability profiles with their status ("supported", "planned" or "unavailable"), file formats, protocols, optional backends and limits. Optional Python backends are only probed when check_backends = TRUE; QuPath is detected from installation files without starting it.

Usage

```
sg_interchange_capabilities(check_backends = FALSE)
```

Arguments

check_backends	Logical; probe optional Python backends through reticulate. Python is initialised only when one has been selected (see the section "Selecting Python" in sg_setup_python()); otherwise they are reported as unavailable.

Details

Profiles: I0 neutral files (manifest, integer TIFF masks, GeoJSON, measurements, integrity inventory); I1 versioned protocols and run envelopes; models model bundles with runtime and checksums; qupath_programs Groovy templates and run.json; stardist_qupath the StarDist extension adapter; control the local control service (segmantR-control-v1); partner_qupflowR and partner_hyperannotatR the partner contracts, which stay planned until a real consumer and QuPath evidence exist.

Value

An sg_capabilities list conforming to capabilities.schema.json, with a digest over its content.

Examples

```
caps <- sg_interchange_capabilities()
caps$profiles$I0$status
caps$schema
```

---

sg_interchange_manifest

Build an interchange manifest

Description

Describes an image, mask or segmentation run with the shared coordinate convention, plane, pixel size, legend, review state, protocol reference, runtime and asset inventory. The result conforms to inst/schema/segmantR-interchange-v1/manifest.schema.json.

Usage

```
sg_interchange_manifest(
  x,
  ...,
  image = NULL,
  assets = list(),
  conversions = list(),
  kind = NULL
)
```

Arguments

x	An sg_image, sg_mask, sg_run, or a named list with image, mask, run and/or measurements.
...	Reserved; must be empty.
image	Optional sg_image the mask belongs to.
assets	Asset records (role, path, media_type, size_bytes, sha256), normally filled by sg_export_interchange().
conversions	Conversion records (from, to, fidelity, notes).
kind	Optional manifest kind override.

Value

An sg_manifest list.

Examples

```
m <- sg_example_mask("fluorescence_nuclei")
man <- sg_interchange_manifest(m)
man$mask$label_count
man$coordinate_convention$y_axis
```

---

sg_load_model *Load a packaged segmantR model*

Description

Reads a .segmantR archive created by sg_package_model() and returns an sg_trained_model object.

Usage

```
sg_load_model(path, expected_digest = NULL, allow_legacy = TRUE)
```

Arguments

path	Character. Path to the .segmantR archive file.
expected_digest	Optional "sha256:<hex>" digest of the archive's checksums.sha256 content (reported as bundle_digest).
allow_legacy	Logical; accept archives without checksums.

Details

Before extracting, entry names are checked (no absolute paths, parent segments, backslashes or drive letters; entry count and total size limits). After extraction, symbolic links are rejected, checksums.sha256 is verified (missing, extra or modified files are errors), the model card schema and major version and the backend are checked. Archives written by segmantR 0.1.0 have no checksums; they load with integrity = "unverified" unless allow_legacy = FALSE.

Value

An sg_trained_model object.

Examples

```
# a dedicated directory with a stand-in weight file
weights <- file.path(tempfile("model"), "cellpose_weights")
dir.create(dirname(weights))
writeBin(as.raw(1:16), weights)
mdl <- new_sg_trained_model(weights, backend = "cellpose",
                            base_model = "cyto3",
                            training_metrics = list(n_epochs = 50L))
archive <- sg_package_model(mdl, tempfile(fileext = ".segmantR"))
model <- sg_load_model(archive)
model$integrity
```

---

sg_mask_legend *Effective legend of a mask*

Description

Returns the stored legend or, when none is stored, a deterministic legend with one row per positive label. Generated object ids are UUID-shaped strings derived from the image binding, the label content and the label value, so the same mask always yields the same ids. Use sg_mask_legend(mask, materialise = TRUE) to store the legend in the mask so ids stay stable through later edits.

Usage

```
sg_mask_legend(mask, materialise = FALSE)
```

Arguments

mask	An sg_mask object.
materialise	Logical; if TRUE, return the mask with the legend stored instead of the legend table.

Value

A tibble with columns label, object_id, class, name, or the updated mask when materialise = TRUE.

Examples

```
labels <- matrix(0L, 6, 6)
labels[2:3, 2:3] <- 1L
labels[5:6, 5:6] <- 2L
sg_mask_legend(new_sg_mask(labels, image_id = "img-1"))
```

sg_mask_review *Stage, review or replace masks*

Description

`sg_stage_mask()` marks a mask as a staged candidate (for example an imported prediction). `sg_review_mask()` records that a human reviewed the current content. `sg_replace_mask()` replaces a mask by a candidate while protecting reviewed data: replacing a reviewed mask requires both `expected_revision` equal to its current revision and `overwrite = TRUE`. Replacing with identical content is a no-op that keeps the current status.

Usage

```
sg_stage_mask(mask, source = NULL, parent = NULL)

sg_review_mask(mask, reviewer = NULL, note = NULL, expected_revision = NULL)

sg_replace_mask(
  current,
  replacement,
  expected_revision = NULL,
  overwrite = FALSE
)
```

Arguments

`mask, current`	An `sg_mask` object.
`source`	Optional short description of where the candidate came from (e.g. `"stardist.2d.v1 prediction"`).
`parent`	Optional `sg_mask` or revision string the candidate derives from.
`reviewer`	Optional reviewer name or identifier.
`note`	Optional free-text note stored in the review history (never evaluated).
`expected_revision`	Revision the caller believes is current. Required to review a staged mask or to replace a reviewed one.
`replacement`	The candidate `sg_mask`.
`overwrite`	Logical; explicit decision to replace a reviewed mask.

Value

The updated `sg_mask`.

Examples

```
m <- new_sg_mask(matrix(c(0L, 1L, 1L, 0L), 2))
staged <- sg_stage_mask(m, source = "example")
rev <- sg_mask_revision(staged)
reviewed <- sg_review_mask(staged, reviewer = "me", expected_revision = rev)
sg_mask_status(reviewed)$status

candidate <- new_sg_mask(matrix(c(1L, 1L, 1L, 0L), 2))
try(sg_replace_mask(reviewed, candidate))
replaced <- sg_replace_mask(reviewed, candidate,
                            expected_revision = rev, overwrite = TRUE)
sg_mask_status(replaced)$status
```

---

sg_mask_revision *Content revision of a mask*

Description

A "sha256:<hex>" digest over the label array (dimensions and row-major little-endian int32 values), the mask type and the effective legend. Identical content always yields the same revision, so repeated imports of the same prediction are idempotent.

Usage

```
sg_mask_revision(mask)
```

Arguments

mask	An sg_mask object.

Value

Character scalar.

Examples

```
m <- new_sg_mask(matrix(c(0L, 1L, 1L, 0L), 2))
sg_mask_revision(m)
```

sg_mask_status *Review state of a mask*

Description

Review state of a mask

Usage

```
sg_mask_status(mask)
```

Arguments

mask	An sg_mask object.

Value

A list with status, revision (current content revision), reviewed_revision, parent_revision and consistent (FALSE when a reviewed mask was modified after review).

Examples

```
sg_mask_status(new_sg_mask(matrix(0L, 3, 3)))
```

sg_mask_to_polygons *Convert mask to polygon geometries*

Description

Extracts cell boundaries from a label mask and returns them as polygon geometries. If the **sf** package is available, returns an sf data frame; otherwise returns a tibble of boundary coordinates.

Usage

```
sg_mask_to_polygons(mask, simplify = TRUE, tolerance = 1)
```

Arguments

mask	An sg_mask object.
simplify	Logical. Simplify polygon geometries? Default TRUE.
tolerance	Numeric. Simplification tolerance in pixels when simplify = TRUE. Default 1.0.

Value

An sf data frame with one row per cell (columns cell_id and geometry) when **sf** is available, or a tibble with columns cell_id, row, col listing boundary coordinates.

Examples

```
labels <- matrix(0L, nrow = 20, ncol = 20)
labels[3:8, 3:8] <- 1L
labels[12:18, 12:18] <- 2L
mask <- new_sg_mask(labels)
polys <- sg_mask_to_polygons(mask, simplify = FALSE)
head(polys)
```

---

sg_merge_masks *Merge nuclear and cell body masks*

Description

Combines a nuclear segmentation mask with a cell body segmentation mask to produce paired nuclear-cell assignments.

Usage

```
sg_merge_masks(nuclear_mask, cell_mask, method = c("assign", "expand"))
```

Arguments

nuclear_mask	An sg_mask object for nuclei.
cell_mask	An sg_mask object for cell bodies.
method	Character. "assign" (default) assigns nuclei to overlapping cells; "expand" expands nuclear labels outward to fill cell boundaries.

Value

A list with elements $nuclear and $cell, both sg_mask objects with matched cell IDs.

Examples

```
nuc_labels <- matrix(0L, nrow = 20, ncol = 20)
nuc_labels[5:7, 5:7] <- 1L
nuc_labels[14:16, 14:16] <- 2L
cell_labels <- matrix(0L, nrow = 20, ncol = 20)
cell_labels[3:9, 3:9] <- 1L
cell_labels[12:18, 12:18] <- 2L
nuc_mask <- new_sg_mask(nuc_labels)
cell_mask <- new_sg_mask(cell_labels)
merged <- sg_merge_masks(nuc_mask, cell_mask)
print(merged$nuclear)
```

sg_model_card *Display a formatted model card*

Description

Prints a human-readable model card for a trained segmentation model and returns the card contents as a tibble.

Usage

```
sg_model_card(trained_model)
```

Arguments

trained_model	An sg_trained_model object.

Value

A tibble with columns field and value representing the model card contents, returned invisibly.

Examples

```
# a dedicated directory with a stand-in weight file
weights <- file.path(tempfile("model"), "cellpose_weights")
dir.create(dirname(weights))
writeBin(as.raw(1:16), weights)
mdl <- new_sg_trained_model(
  model_path = weights,
  backend = "cellpose",
  base_model = "cyto3",
  training_metrics = list(n_epochs = 100L),
  model_card = list(author = "Test User", tissue = "lung")
)
sg_model_card(mdl)
```

sg_package_model *Package a trained model for sharing*

Description

Creates a .segmantR archive (ZIP format) containing model weights, a model_card.json, and optionally training data.

Usage

```
sg_package_model(
  trained_model,
  output_path,
  name = NULL,
  description = NULL,
  include_training_data = FALSE,
  format = c("segmantR", "cellpose", "both"),
  embed_weights = TRUE,
  license = NULL,
  overwrite = TRUE
)
```

Arguments

trained_model	An sg_trained_model object.
output_path	Character. Path for the output archive file.
name	Character or NULL. Human-readable model name.
description	Character or NULL. Short description of the model.
include_training_data	Logical. Include training data in the archive? Default FALSE. When TRUE, the dataset manifest (not the pixels) is included; tiles stay in their own integrity-checked dataset directory.
format	Character. Archive format: "segmantR" (default), "cellpose", or "both".
embed_weights	Logical; copy weights into the archive (default) or record them as a hashed reference relative to the archive directory.
license	Optional license list (id, name, source, text); defaults to trained_model$license.
overwrite	Logical; replace an existing archive.

Details

The archive keeps relative paths and contains a versioned model_card.json (schema segmantR-model-card-v1, still carrying the fields written by segmantR 0.1.0), runtime.json, dataset_manifest.json when the model has one, license.json when a license is given, the weights under weights/ (or a hashed relative reference when embed_weights = FALSE) and checksums.sha256 covering every file. Python environments are never copied into a bundle.

Value

The output file path, returned invisibly.

Examples

```
# a dedicated directory with a stand-in weight file
weights <- file.path(tempfile("model"), "cellpose_weights")
dir.create(dirname(weights))
writeBin(as.raw(1:16), weights)
```

```
mdl <- new_sg_trained_model(
  model_path = weights,
  backend = "cellpose",
  base_model = "cyto3",
  training_metrics = list(n_epochs = 50L)
)
out <- sg_package_model(mdl, tempfile(fileext = ".segmantR"))
```

`sg_plot_comparison` *Side-by-side comparison of two masks*

Description

Shows two segmentation masks (and optionally the source image) next to each other using faceted panels.

Usage

```
sg_plot_comparison(
  mask_a,
  mask_b,
  image = NULL,
  labels = c("Method A", "Method B")
)
```

Arguments

`mask_a`	An `sg_mask` object (left panel).
`mask_b`	An `sg_mask` object (right panel).
`image`	An `sg_image` object, or `NULL`.
`labels`	Character vector of length 2 giving panel titles (default `c("Method A", "Method B")`).

Value

A `ggplot2` object.

Examples

```
m1 <- new_sg_mask(matrix(c(0L, 1L, 1L, 0L), 2, 2))
m2 <- new_sg_mask(matrix(c(0L, 0L, 1L, 1L), 2, 2))
p <- sg_plot_comparison(m1, m2)
p
```

`sg_plot_features` *Scatter plot of cell features*

Description

Produces a scatter plot of two numeric cell features, with optional colour and faceting variables.

Usage

```
sg_plot_features(
  features,
  x = "area",
  y = "mean_intensity",
  color = NULL,
  facet = NULL
)
```

Arguments

`features`	A data frame of cell features (e.g., from `sg_compute_features()`).
`x`	Character, column name for the x-axis (default `"area"`).
`y`	Character, column name for the y-axis (default `"mean_intensity"`).
`color`	Character column name for point colour, or `NULL`.
`facet`	Character column name for faceting, or `NULL`.

Value

A `ggplot2` object.

Examples

```
feat <- data.frame(
  cell_id = 1:10,
  area = rpois(10, 200),
  mean_intensity = runif(10)
)
p <- sg_plot_features(feat, x = "area", y = "mean_intensity")
p
```

sg_plot_mask *Render a mask as a coloured image*

Description

Assigns colours to cell regions based on cell ID, area, circularity, or cluster membership.

Usage

```
sg_plot_mask(
  mask,
  color_by = c("id", "area", "circularity", "cluster"),
  features = NULL,
  palette = "viridis"
)
```

Arguments

mask	An sg_mask object.
color_by	Character, one of "id", "area", "circularity", or "cluster".
features	A data frame of cell features (required when color_by is "area", "circularity", or "cluster"). Must contain a cell_id column.
palette	Character, name of a viridis palette (default "viridis").

Value

A ggplot2 object.

Examples

```
labels <- matrix(c(0L, 1L, 1L, 2L, 2L, 0L, 3L, 3L, 0L), 3, 3)
mask <- new_sg_mask(labels)
p <- sg_plot_mask(mask)
p
```

sg_plot_metrics *Bar chart of evaluation metrics*

Description

Displays a bar chart of segmentation evaluation metrics such as IoU, precision, recall, and F1.

Usage

```
sg_plot_metrics(eval_result)
```

Arguments

eval_result	A data frame with columns metric and value, e.g., from sg_evaluate().

Value

A ggplot2 object.

Examples

```
res <- data.frame(
  metric = c("IoU", "Precision", "Recall", "F1"),
  value = c(0.82, 0.90, 0.85, 0.87)
)
p <- sg_plot_metrics(res)
p
```

---

sg_plot_overlay *Overlay cell boundaries on an image*

Description

Renders a single image channel as a raster and overlays cell outlines from a segmentation mask.

Usage

```
sg_plot_overlay(
  image,
  mask = NULL,
  channel = 1L,
  outline_color = "#FF00FF",
  outline_width = 1,
  fill_alpha = 0,
  label_cells = FALSE,
  highlight = NULL
)
```

Arguments

image	An sg_image object.
mask	An sg_mask object, or NULL to show the image only.
channel	Integer, which channel to display (default 1L).
outline_color	Character, colour for cell outlines (default "#FF00FF").
outline_width	Numeric, width of the outline strokes (default 1).
fill_alpha	Numeric in [0, 1], fill opacity for cell regions (default 0).
label_cells	Logical, whether to draw cell-ID labels at centroids (default FALSE).
highlight	Integer vector of cell IDs to highlight, or NULL for all cells.

Value

A ggplot2 object.

Examples

```
img <- new_sg_image(matrix(runif(400), 20, 20))
p <- sg_plot_overlay(img)
p
```

`sg_predict_model` *Predict with a trained model*

Description

Backend-agnostic prediction facade. The model's backend selects the protocol (`stardist.2d.v1` or `cellpose.2d.v1`); the run goes through `sg_protocol_run()` so parameters are validated and recorded, the backend is checked at the point of use, and the resulting mask is **staged** with the model digest in its provenance. Predictions never overwrite reviewed masks; use `sg_replace_mask()` to adopt a reviewed result explicitly.

Usage

```
sg_predict_model(model, image, ..., protocol = NULL, output = c("mask", "run"))
```

Arguments

`model`	An `sg_trained_model` or the path of a `.segmantR` archive (loaded with integrity verification).
`image`	An `sg_image`.
`...`	Protocol parameters (see `sg_protocol_schema(protocol)`).
`protocol`	Optional protocol id overriding the backend default.
`output`	`"mask"` or `"run"` (see `sg_protocol_run()`).

Value

A staged `sg_mask` or an `sg_run`.

Examples

```
mdl <- new_sg_trained_model(tempfile("model"), backend = "stardist",
                            base_model = "none", training_metrics = list())
img <- sg_example_image("fluorescence_nuclei")
run <- sg_predict_model(mdl, img, output = "run")
# "unavailable" unless a Python with 'stardist' has been selected
# (see sg_setup_python())
run$record$status
```

`sg_prepare_training_data` *Prepare training data tiles*

Description

Builds the dataset manifest with `sg_dataset_manifest()` and materialises the included tiles: image tiles as float64 TIFF (exact values, `[y, x, channel]` pages) and mask tiles as unsigned integer TIFF with labels renumbered 1..N per tile in raster order. Tile object lists map tile labels to source object ids. With a `destination`, a directory with `dataset_manifest.json`, `tiles/<split>/images|masks/*.tif` and `integrity.json` is written; otherwise tiles are returned in memory.

Usage

```
sg_prepare_training_data(pairs, destination = NULL, ..., overwrite = FALSE)
```

Arguments

`pairs`	A list of training pairs. Each element is a list with `image` (`sg_image`), `mask` (`sg_mask`) and the grouping ids `subject_id`, `sample_id` and/or `slide_id`.
`destination`	Optional output directory (must not exist or be empty).
`...`	Further arguments for `sg_dataset_manifest()`.
`overwrite`	Logical; replace an existing prepared dataset.

Value

An `sg_training_set` with `manifest`, `path` (or `NULL`) and `tiles` (in-memory list when `destination` is `NULL`).

Examples

```
img <- sg_example_image("fluorescence_nuclei")
# The synthetic example mask contains overlapping ellipses; split them
# into separate instances first.
msk <- sg_cleanup_labels(sg_example_mask("fluorescence_nuclei"),
                         disconnected = "split")
ts <- sg_prepare_training_data(
  list(list(image = img, mask = msk, subject_id = "A")),
  tile_size = 32L, min_objects = 0L,
  split = c(train = 1, validation = 0, test = 0)
)
length(ts$tiles)
```

sg_preprocess *Preprocess an image*

Description

Applies a chain of preprocessing steps to an sg_image object, including optional stain normalisation, denoising, and contrast enhancement.

Usage

```
sg_preprocess(
  image,
  stain_normalize = c("none", "macenko", "reinhard", "vahadane"),
  denoise = FALSE,
  contrast = c("none", "clahe", "stretch"),
  target_resolution = NULL
)
```

Arguments

image	An sg_image object.
stain_normalize	Character string specifying the stain normalisation method. One of "none" (default), "macenko", "reinhard", or "vahadane".
denoise	Logical; if TRUE, a simple mean filter is applied for denoising. Default is FALSE.
contrast	Character string specifying contrast enhancement. One of "none" (default), "clahe", or "stretch".
target_resolution	Finite positive numeric scalar or NULL. If supplied, resample to this many microns per current array pixel using bilinear interpolation. Requested resampling of anisotropic pixels is unsupported and raises an error; leave NULL to retain them unchanged. Both pixel sizes must be finite and positive to resample. Partial or invalid calibration raises an error; if both sizes are unknown, the existing message-and-skip behaviour is retained.

Value

A new sg_image object with its history field updated to record the preprocessing steps applied.

Examples

```
pixels <- matrix(runif(400), nrow = 20, ncol = 20)
img <- new_sg_image(pixels)
img2 <- sg_preprocess(img)
img3 <- sg_preprocess(img, contrast = "stretch", denoise = TRUE)
```

sg_protocol_get *Get a registered protocol*

Description

Get a registered protocol

Usage

```
sg_protocol_get(id, version = NULL)
```

Arguments

id	Protocol identifier, e.g. "threshold.otsu.v1".
version	Optional exact semantic version. NULL returns the registered version.

Value

An sg_protocol object (a validated named list).

Examples

```
p <- sg_protocol_get("threshold.otsu.v1")
p$version
```

sg_protocol_list *List registered segmentation protocols*

Description

List registered segmentation protocols

Usage

```
sg_protocol_list(check_available = FALSE)
```

Arguments

check_available	Logical; if TRUE, probe optional Python backends through reticulate; Python is initialised only when one has been selected (see the section "Selecting Python" in sg_setup_python()). Default FALSE reports NA availability for optional backends.

Value

A tibble with one row per protocol: `id`, `version`, `title`, `family`, `backend`, `delegate`, `requires_python`, `status` (`"core"` or `"optional"`), `available`, `qupath_mapping` and `digest`.

Examples

```
sg_protocol_list()
```

---

sg_protocol_run *Run a declarative segmentation protocol*

Description

Validates the protocol, the parameters and the inputs, delegates to the corresponding direct function (`sg_segment_threshold()`, `sg_segment_watershed()`, `sg_segment_propagate()`, `sg_cleanup_labels()`, `sg_segment_stardist()`, `sg_segment_cellpose()`, `sg_segment_mesmer()`) with an explicit, recorded argument mapping, and returns the mask or a run envelope.

Usage

```
sg_protocol_run(image, protocol, ..., output = c("mask", "run"))
```

Arguments

`image`	The primary input: an `sg_image`, or an `sg_mask` for mask-input protocols such as `postprocess.label-cleanup.v1`.
`protocol`	Protocol id, `sg_protocol`, named list, JSON text or `.json` file path.
`...`	Parameter overrides (for example `min_area = 20L`) and additional declared inputs (for example `seeds = nuclei_mask` for `propagate.voronoi.v1`, `model = trained_model`). Unknown names are errors.
`output`	`"mask"` (default) returns the `sg_mask`; `"run"` returns an `sg_run` with `mask`, `measurements` and a schema-valid `record`.

Details

Core protocols run without Python. Optional DNN protocols check their backend first; when it is missing, `output = "mask"` signals an `sg_capability_error` and `output = "run"` returns a run with `status = "unavailable"`. There is never a silent fallback to another method. DNN predictions are returned as staged masks.

Value

An `sg_mask` or an `sg_run`.

See Also

`sg_protocol_list()`, `sg_protocol_schema()`, `sg_protocol_validate()`, `sg_export_interchange()`

Examples

```
img <- sg_example_image("fluorescence_nuclei")
mask <- sg_protocol_run(img, "threshold.otsu.v1", min_area = 5L)
run <- sg_protocol_run(img, "threshold.otsu.v1", min_area = 5L,
                       output = "run")
run$record$delegate$arguments
identical(mask$labels, run$mask$labels)
```

---

sg_protocol_schema *Protocol schema and parameter specifications*

Description

Protocol schema and parameter specifications

Usage

```
sg_protocol_schema(id = NULL)
```

Arguments

id	NULL for the JSON Schema of segmantR-protocol-v1 (as a parsed list), or a protocol id for its parameter table.

Value

For id = NULL, the parsed JSON Schema. Otherwise a tibble with name, type, default, nullable, minimum, maximum, enum, unit, requires_calibration, maps_to and description.

Examples

```
names(sg_protocol_schema()$properties)
sg_protocol_schema("threshold.otsu.v1")[, c("name", "default", "unit")]
```

---

sg_protocol_validate *Validate a protocol, optionally against an image and parameters*

Description

Checks the definition against segmantR-protocol-v1 (schema family and major version, JSON Schema, parameter mapping completeness, defaults), resolves parameters (unknown names and out-of-range values are errors), and, when an image is given, checks the input contract: array order and size, value semantics, channel/band selection and pixel calibration.

Usage

```
sg_protocol_validate(protocol, image = NULL, ..., error = TRUE)
```

Arguments

protocol	Protocol id, sg_protocol, named list, JSON text or path to a .json file.
image	Optional sg_image (or sg_mask for mask-input protocols).
...	Parameter overrides and additional inputs (for example seeds = mask).
error	Logical; if FALSE, return a report with ok = FALSE instead of signalling the first error.

Value

An sg_protocol_validation list with ok, protocol, parameters, checks (tibble) and error (condition or NULL).

Examples

```
img <- sg_example_image("fluorescence_nuclei")
v <- sg_protocol_validate("threshold.otsu.v1", img, min_area = 5L)
v$ok
bad <- sg_protocol_validate("threshold.otsu.v1", img, min_area = -1L,
                            error = FALSE)
bad$error$code
```

---

sg_read_envi *Read an ENVI cube window into an sg_image*

Description

Reads only the requested bands and pixel window by seeking in the data file (BSQ, BIL or BIP; either byte order). Values are returned as stored: gain, offset and reflectance scale factors are recorded but never applied, and value_semantics is whatever the caller declares (default "unknown"), so raw counts, reflectance, radiance, intensity and absorbance are never mixed silently.

Usage

```
sg_read_envi(
  path,
  bands = NULL,
  window = NULL,
  value_semantics = "unknown",
  nodata_to_na = TRUE,
  source_digest = c("full", "header", "none"),
  data_file = NULL
)
```

Arguments

path	Path to the .hdr file or the data file.
bands	Optional 1-based band indices to read (default all).
window	Optional c(row_min, row_max, col_min, col_max) (1-based, inclusive).
value_semantics	Declared meaning of the stored values.
nodata_to_na	Replace the header data ignore value by NA.
source_digest	"full" hashes header and data file (identity of the source), "header" hashes only the header, "none" skips hashing.
data_file	Optional explicit data file path.

Value

An sg_image with pixels [y, x, band], a band table, origin set to the window offset, calibration_digest, and metadata$read_accounting (bytes_read, bands_read).

Examples

```
d <- tempfile("envi")
dir.create(d)
writeLines(c("ENVI", "samples = 3", "lines = 2", "bands = 2",
             "data type = 12", "interleave = bip", "byte order = 0",
             "wavelength = {500, 600}"), file.path(d, "cube.hdr"))
writeBin(1:12, file.path(d, "cube"), size = 2L, endian = "little")
img <- sg_read_envi(file.path(d, "cube.hdr"), bands = 2,
                    value_semantics = "raw")
img$pixels
```

---

sg_read_field *Read a square multichannel field around a point*

Description

Converts a centre in physical units (for example micrometres) to pixels and reads a square window of round(size / pixel_size) pixels in each requested channel. The top-left pixel is round(centre / pixel_size) - n %/% 2, so a field is reproducible from its centre, size and pixel size alone.

Usage

```
sg_read_field(path, channels, centre, size, pixel_size = 1, fill = 0)
```

Arguments

path	Path to the image file.
channels	Channel names or numbers.
centre	Numeric c(x, y) in physical units, or a data frame whose first row has columns x and y.
size	Field width in physical units.
pixel_size	Physical size of one pixel.
fill	Value used outside the image.

Value

A list of class sg_field with channels (named list of n x n matrices), x0, y0, n, pixel_size, centre, size and extent (c(xmin, xmax, ymin, ymax) of the pixel edges in physical units).

See Also

sg_read_fields(), sg_read_window()

Examples

```
f <- tempfile(fileext = ".tif")
planes <- list(matrix(1:100, 10, 10), matrix(101:200, 10, 10))
invisible(tiff::writeTIFF(lapply(planes, function(p) p / 255), f,
                          bits.per.sample = 8L))
fld <- sg_read_field(f, c("ch1", "ch2"), centre = c(2.5, 2.5), size = 2,
                     pixel_size = 0.5)
fld$channels$ch2
```

---

sg_read_fields *Read many square fields, for example cell galleries*

Description

Reads one sg_read_field() per row of a table of centres. Each row names its image through source, looked up in paths.

Usage

```
sg_read_fields(
  centres,
  paths,
  channels = character(0),
  size,
  pixel_size = 1,
  fill = 0,
```

```
  x = "x",
  y = "y",
  source = NULL,
  id = NULL,
  channel_columns = NULL,
  key_prefix = ""
)
```

Arguments

centres	Data frame with centre coordinates and the image key.
paths	Named character vector or list: image key to file path, or a single path used for all rows when source is NULL.
channels	Channel names or numbers read for every field.
size, pixel_size, fill	As in sg_read_field().
x, y, source	Column names of the centre coordinates and image key.
id	Optional column used to name the fields.
channel_columns	Optional columns of centres naming further channels to read for each row (for example a row-specific marker).
key_prefix	Prefix added to the source values before looking them up in paths.

Value

A named list of sg_field objects in row order; each carries the table row (as a one-row data frame) in row.

See Also

sg_read_field()

Examples

```
f <- tempfile(fileext = ".tif")
planes <- list(matrix(1:100, 10, 10), matrix(101:200, 10, 10))
invisible(tiff::writeTIFF(lapply(planes, function(p) p / 255), f,
                          bits.per.sample = 8L))
cells <- data.frame(id = c("a", "b"), x = c(2.5, 4), y = c(2.5, 3))
fields <- sg_read_fields(cells, f, channels = c("ch1", "ch2"), size = 2,
                         pixel_size = 0.5, id = "id")
fields$b$channels$ch2
```

sg_read_image *Read an image file*

Description

Reads TIFF, PNG, or JPEG image files and returns an `sg_image` object. Dispatches to `EBImage::readImage()` when available, otherwise uses `imager::load.image()`. The returned pixel array is always ordered `[y, x, channel]`, and the object records the SHA-256 digest of the file in `source_digest` plus its base name (never the full path) in `metadata$source_name`.

Usage

```
sg_read_image(path, ...)
```

Arguments

`path`	Character string, path to the image file.
`...`	Additional arguments passed to the underlying reader.

Value

An `sg_image` object.

Examples

```
f <- tempfile(fileext = ".tif")
EBImage::writeImage(EBImage::Image(matrix(seq(0, 1, length.out = 200), 20, 10)),
                    f)
img <- sg_read_image(f)
dim(img$pixels)
```

sg_read_window *Read a rectangular pixel window of one channel*

Description

Reads `height` rows and `width` columns of one channel plane starting at the 0-based pixel offset (`x0`, `y0`). Only the image blocks that cover the window are decoded, so windows of very large whole-slide files are cheap. Parts of the window outside the image are filled with `fill`, which makes fields at the slide edge keep their nominal size.

Usage

```
sg_read_window(path, channel, x0, y0, width, height = width, fill = 0)
```

Arguments

path	Path to the image file.
channel	Channel number (as in sg_image_channels()) or channel name.
x0, y0	0-based column and row of the top-left pixel; may be negative.
width, height	Window size in pixels.
fill	Value used outside the image.

Value

A numeric matrix with height rows and width columns ([y, x]), with attributes x0, y0 and channel (the name).

See Also

sg_read_field(), sg_image_channels()

Examples

```
f <- tempfile(fileext = ".tif")
plane <- matrix(0:11, nrow = 3, byrow = TRUE)
invisible(tiff::writeTIFF(plane / 255, f, bits.per.sample = 8L))
sg_read_window(f, 1, x0 = 1, y0 = 0, width = 2, height = 2)
```

---

sg_roi *Histology ROI objects*

Description

ROI geometries use null-CRS image coordinates: top-left origin, x right, y down. Layers preserve drawing order and projects retain edit provenance. These classes are distinct from segmantR's instance-mask classes.

See Also

sg_roi_rect(), sg_roi_frame(), sg_roi_layer(), sg_roi_project(), sg_roi_mask()

sg_roi_add_layer *Add a layer to a project*

Description

Add a layer to a project

Usage

```
sg_roi_add_layer(project, layer, call = rlang::caller_env())
```

Arguments

project	An sg_roi_project.
layer	An sg_roi_layer with a name not already used in the project.
call	The calling environment, for error reporting.

Value

A new sg_roi_project with the layer added. The input is unchanged.

See Also

Other projects: sg_roi_add_roi(), sg_roi_layers(), sg_roi_project(), sg_roi_remove_layer(), sg_roi_remove_roi(), sg_roi_rois(), sg_roi_summary(), sg_roi_validate()

Examples

```
proj <- sg_roi_project(sg_roi_frame(c(20, 20)))
proj <- sg_roi_add_layer(proj, sg_roi_layer("tissue"))
names(proj$layers)
```

sg_roi_add_roi *Add an ROI to a project layer*

Description

Add an ROI to a project layer

Usage

```
sg_roi_add_roi(project, layer, roi, call = rlang::caller_env())
```

Arguments

project	An sg_roi_project.
layer	Single string name of the target layer.
roi	An sg_roi.
call	The calling environment, for error reporting.

Value

A new sg_roi_project with the ROI added to the layer. A colliding ROI identifier is re-minted to keep identifiers unique across the project. The input is unchanged.

See Also

Other projects: sg_roi_add_layer(), sg_roi_layers(), sg_roi_project(), sg_roi_remove_layer(), sg_roi_remove_roi(), sg_roi_rois(), sg_roi_summary(), sg_roi_validate()

Examples

```
proj <- sg_roi_project(sg_roi_frame(c(20, 20)), sg_roi_layer("tissue"))
proj <- sg_roi_add_roi(proj, "tissue", sg_roi_rect(2, 2, 8, 8, label = "tissue"))
sg_roi_rois(proj)$area_px
```

---

sg_roi_area *ROI area*

Description

ROI area

Usage

```
sg_roi_area(
  roi,
  level = 0L,
  units = c("px", "physical"),
  call = rlang::caller_env()
)
```

Arguments

roi	An sg_roi.
level	Integer pyramid level at which to express the area. Default 0.
units	Either "px" (pixels squared, the default) or "physical". Physical units require a pixel_size attribute on the ROI.
call	The calling environment, for error reporting.

Value

A numeric scalar area, or `NA_real_` for non-areal geometry (points, lines).

See Also

Other rois: `sg_roi_bbox()`, `sg_roi_centroid()`, `sg_roi_circle()`, `sg_roi_ellipse()`, `sg_roi_freehand()`, `sg_roi_from_geojson()`, `sg_roi_from_sf()`, `sg_roi_point()`, `sg_roi_polygon()`, `sg_roi_rect()`

Examples

```
sg_roi_area(sg_roi_rect(0, 0, 10, 5, label = "a"))
```

---

sg_roi_bbox *ROI bounding box*

Description

ROI bounding box

Usage

```
sg_roi_bbox(roi, call = rlang::caller_env())
```

Arguments

`roi`	An sg_roi.
`call`	The calling environment, for error reporting.

Value

A numeric vector `c(xmin, ymin, xmax, ymax)` in the ROI's stored coordinates.

See Also

Other rois: `sg_roi_area()`, `sg_roi_centroid()`, `sg_roi_circle()`, `sg_roi_ellipse()`, `sg_roi_freehand()`, `sg_roi_from_geojson()`, `sg_roi_from_sf()`, `sg_roi_point()`, `sg_roi_polygon()`, `sg_roi_rect()`

Examples

```
sg_roi_bbox(sg_roi_circle(50, 50, 10, label = "a"))
```

sg_roi_buffer *Buffer an ROI*

Description

Buffer an ROI

Usage

```
sg_roi_buffer(roi, dist, call = rlang::caller_env())
```

Arguments

roi	An sg_roi.
dist	Numeric buffer distance in pixels (may be negative to erode).
call	The calling environment, for error reporting.

Value

An sg_roi with the buffered geometry.

See Also

Other geometry: sg_roi_check_containment(), sg_roi_check_geometry(), sg_roi_clamp(), sg_roi_contains(), sg_roi_distance(), sg_roi_fix_geometry(), sg_roi_flip_y(), sg_roi_overlaps(), sg_roi_rescale(), sg_roi_ring(), sg_roi_rois_overlap(), sg_roi_setops, sg_roi_simplify(), sg_roi_snap(), sg_roi_transform()

Examples

```
sg_roi_buffer(sg_roi_point(10, 10, label = "a"), dist = 5)
```

sg_roi_centroid *ROI centroid*

Description

ROI centroid

Usage

```
sg_roi_centroid(roi, call = rlang::caller_env())
```

Arguments

roi	An sg_roi.
call	The calling environment, for error reporting.

Value

A numeric vector c(x, y) in the ROI's stored coordinates (at its own level).

See Also

Other rois: sg_roi_area(), sg_roi_bbox(), sg_roi_circle(), sg_roi_ellipse(), sg_roi_freehand(), sg_roi_from_geojson(), sg_roi_from_sf(), sg_roi_point(), sg_roi_polygon(), sg_roi_rect()

Examples

```
sg_roi_centroid(sg_roi_rect(0, 0, 10, 10, label = "a"))
```

---

sg_roi_check_containment

Check cross-layer containment

Description

Report ROIs in a constrained layer that fall outside their container region. A layer declares the constraint through its metadata, e.g. sg_roi_layer("state", within = list(layer = "anatomy", label = "wound")); then every ROI in state must be spatially covered by the union of the anatomy layer's wound ROIs. This enforces the annotation rule that a class is painted only inside its region of interest. The report mirrors the shape of sg_roi_check_geometry().

Usage

```
sg_roi_check_containment(project, tol = 0, call = rlang::caller_env())
```

Arguments

project	An sg_roi_project.
tol	Non-negative slack, in level-0 pixels, by which the container is grown before testing, to tolerate sub-pixel tracing overhang. Default 0.
call	The calling environment, for error reporting.

Value

A tibble::tibble with columns roi_id, layer, issue, severity ("warning" for an ROI outside its container, "error" for a missing/empty container), and fixable (always FALSE). A 0-row tibble when every constrained ROI is contained, or when no layer declares a within constraint.

See Also

sg_roi_check_geometry(), sg_roi_validate()

Other geometry: sg_roi_buffer(), sg_roi_check_geometry(), sg_roi_clamp(), sg_roi_contains(), sg_roi_distance(), sg_roi_fix_geometry(), sg_roi_flip_y(), sg_roi_overlaps(), sg_roi_rescale(), sg_roi_ring(), sg_roi_rois_overlap(), sg_roi_setops, sg_roi_simplify(), sg_roi_snap(), sg_roi_transform()

Examples

```
anatomy <- sg_roi_layer_add(sg_roi_layer("anatomy", labels = "wound"),
                        sg_roi_rect(2, 2, 8, 8, label = "wound"))
state <- sg_roi_layer("state", labels = "injury",
                  within = list(layer = "anatomy", label = "wound"))
state <- sg_roi_layer_add(state, sg_roi_rect(0, 0, 4, 4, label = "injury"))
sg_roi_check_containment(sg_roi_project(sg_roi_frame(c(100, 100)), list(anatomy, state)))
```

---

sg_roi_check_geometry *Check the geometry of every ROI in a project*

Description

Check the geometry of every ROI in a project

Usage

```
sg_roi_check_geometry(project, call = rlang::caller_env())
```

Arguments

project	An sg_roi_project.
call	The calling environment, for error reporting.

Value

A tibble::tibble with columns roi_id (character), issue (character), severity (character, "error"/"warning"), and fixable (logical). Detected issues: self-intersection, invalid geometry, zero area, degenerate rings, out-of-bounds vertices, duplicate consecutive vertices, ring orientation, and NA coordinates. A 0-row tibble when all geometry is valid.

See Also

sg_roi_fix_geometry(), sg_roi_validate()

Other geometry: sg_roi_buffer(), sg_roi_check_containment(), sg_roi_clamp(), sg_roi_contains(), sg_roi_distance(), sg_roi_fix_geometry(), sg_roi_flip_y(), sg_roi_overlaps(), sg_roi_rescale(), sg_roi_ring(), sg_roi_rois_overlap(), sg_roi_setops, sg_roi_simplify(), sg_roi_snap(), sg_roi_transform()

Examples

```
lyr <- sg_roi_layer_add(sg_roi_layer("tissue"),
                        sg_roi_rect(2, 2, 8, 8, label = "tissue"))
proj <- sg_roi_project(sg_roi_frame(c(20, 20)), lyr)
sg_roi_check_geometry(proj)
```

`sg_roi_circle` *Create a circular ROI*

Description

The circle is approximated by a regular polygon of `n_seg` segments.

Usage

```
sg_roi_circle(
  x,
  y,
  r,
  label,
  level = 0L,
  n_seg = 64L,
  ...,
  call = rlang::caller_env()
)
```

Arguments

`x, y`	Numeric centre coordinates in image pixels.
`r`	Positive numeric radius in pixels.
`label`	Single string class label.
`level`	Integer pyramid level the coordinates refer to. Default `0`.
`n_seg`	Integer number of polygon segments approximating the circle.
`...`	Additional named attributes stored on the ROI. The reserved names `author`, `source`, and `id` set those fields; all others become user attributes.
`call`	The calling environment, for error reporting.

Value

An sg_roi with `POLYGON` geometry and `attributes$shape = "circle"`, so exports can flag the polygon as an approximation.

See Also

Other rois: `sg_roi_area()`, `sg_roi_bbox()`, `sg_roi_centroid()`, `sg_roi_ellipse()`, `sg_roi_freehand()`, `sg_roi_from_geojson()`, `sg_roi_from_sf()`, `sg_roi_point()`, `sg_roi_polygon()`, `sg_roi_rect()`

Examples

```
r <- sg_roi_circle(50, 50, r = 10, label = "nucleus")
```

`sg_roi_clamp` *Clip an ROI to image bounds*

Description

Clip an ROI to image bounds

Usage

```
sg_roi_clamp(roi, dims, call = rlang::caller_env())
```

Arguments

`roi`	An sg_roi.
`dims`	Integer `c(width, height)` bounds in pixels.
`call`	The calling environment, for error reporting.

Value

An sg_roi clipped to `[0, width] x [0, height]`. Warns if the clip empties the geometry.

See Also

Other geometry: `sg_roi_buffer()`, `sg_roi_check_containment()`, `sg_roi_check_geometry()`, `sg_roi_contains()`, `sg_roi_distance()`, `sg_roi_fix_geometry()`, `sg_roi_flip_y()`, `sg_roi_overlaps()`, `sg_roi_rescale()`, `sg_roi_ring()`, `sg_roi_rois_overlap()`, `sg_roi_setops`, `sg_roi_simplify()`, `sg_roi_snap()`, `sg_roi_transform()`

Examples

```
sg_roi_clamp(sg_roi_rect(-5, -5, 20, 20, label = "a"), dims = c(10, 10))
```

sg_roi_contains *Test whether points fall inside an ROI*

Description

Test whether points fall inside an ROI

Usage

```
sg_roi_contains(roi, x, y, call = rlang::caller_env())
```

Arguments

roi	An sg_roi.
x, y	Numeric coordinate vectors of equal length.
call	The calling environment, for error reporting.

Value

A logical vector of length `length(x)`.

See Also

Other geometry: sg_roi_buffer(), sg_roi_check_containment(), sg_roi_check_geometry(), sg_roi_clamp(), sg_roi_distance(), sg_roi_fix_geometry(), sg_roi_flip_y(), sg_roi_overlaps(), sg_roi_rescale(), sg_roi_ring(), sg_roi_rois_overlap(), sg_roi_setops, sg_roi_simplify(), sg_roi_snap(), sg_roi_transform()

Examples

```
sg_roi_contains(sg_roi_rect(0, 0, 10, 10, label = "a"), c(5, 20), c(5, 20))
```

sg_roi_dims *Image dimensions at a pyramid level*

Description

Return the pixel dimensions `c(width, height)` of an sg_roi_frame at a given pyramid level.

Usage

```
sg_roi_dims(x, level = 0L, call = rlang::caller_env())
```

Arguments

x	An sg_roi_frame.
level	Integer pyramid level, 0 for full resolution. Must be in the declared zero-based pyramid range.
call	The calling environment, for error reporting.

Value

An integer vector of length 2, c(width, height).

See Also

sg_roi_frame()

Examples

```
sg_roi_dims(sg_roi_frame(c(100, 80)))
```

---

sg_roi_distance *Distance between two ROIs*

Description

Distance between two ROIs

Usage

```
sg_roi_distance(roi1, roi2, call = rlang::caller_env())
```

Arguments

roi1, roi2	sg_roi objects.
call	The calling environment, for error reporting.

Value

A numeric scalar distance in level-0 pixels.

See Also

Other geometry: sg_roi_buffer(), sg_roi_check_containment(), sg_roi_check_geometry(), sg_roi_clamp(), sg_roi_contains(), sg_roi_fix_geometry(), sg_roi_flip_y(), sg_roi_overlaps(), sg_roi_rescale(), sg_roi_ring(), sg_roi_rois_overlap(), sg_roi_setops, sg_roi_simplify(), sg_roi_snap(), sg_roi_transform()

Examples

```
a <- sg_roi_rect(0, 0, 4, 4, label = "a")
b <- sg_roi_rect(10, 0, 14, 4, label = "b")
sg_roi_distance(a, b)
```

`sg_roi_ellipse` *Create an elliptical ROI*

Description

Create an elliptical ROI

Usage

```
sg_roi_ellipse(
  x,
  y,
  rx,
  ry,
  rotation = 0,
  label,
  level = 0L,
  n_seg = 64L,
  ...,
  call = rlang::caller_env()
)
```

Arguments

`x`, `y`	Numeric centre coordinates in image pixels.
`rx`, `ry`	Positive numeric semi-axis lengths in pixels.
`rotation`	Rotation of the major axis in degrees, clockwise in image orientation. Default `0`.
`label`	Single string class label.
`level`	Integer pyramid level the coordinates refer to. Default `0`.
`n_seg`	Integer number of polygon segments approximating the ellipse.
`...`	Additional named attributes stored on the ROI. The reserved names `author`, `source`, and `id` set those fields; all others become user attributes.
`call`	The calling environment, for error reporting.

Value

An sg_roi with `POLYGON` geometry and `attributes$shape = "ellipse"`, so exports can flag the polygon as an approximation.

See Also

Other rois: sg_roi_area(), sg_roi_bbox(), sg_roi_centroid(), sg_roi_circle(), sg_roi_freehand(), sg_roi_from_geojson(), sg_roi_from_sf(), sg_roi_point(), sg_roi_polygon(), sg_roi_rect()

Examples

```
r <- sg_roi_ellipse(50, 50, rx = 20, ry = 10, rotation = 30, label = "cell")
```

---

sg_roi_fix_geometry *Repair the geometry of a project*

Description

Apply sf::st_make_valid() to invalid ROIs (which also drops duplicate consecutive vertices) and clamp out-of-bounds vertices into the image bounds, reporting what changed.

Usage

```
sg_roi_fix_geometry(project, verbose = TRUE, call = rlang::caller_env())
```

Arguments

project	An sg_roi_project.
verbose	Logical; whether to report repairs. Default TRUE.
call	The calling environment, for error reporting.

Value

The sg_roi_project with repaired geometry. Repairs are reported via cli::cli_inform() when verbose is TRUE. Issues that cannot be repaired (zero area, degenerate, NA coordinates) are left in place and reported.

See Also

sg_roi_check_geometry()

Other geometry: sg_roi_buffer(), sg_roi_check_containment(), sg_roi_check_geometry(), sg_roi_clamp(), sg_roi_contains(), sg_roi_distance(), sg_roi_flip_y(), sg_roi_overlaps(), sg_roi_rescale(), sg_roi_ring(), sg_roi_rois_overlap(), sg_roi_setops, sg_roi_simplify(), sg_roi_snap(), sg_roi_transform()

Examples

```
# a rectangle reaching past the image edge is clamped into the image
lyr <- sg_roi_layer_add(sg_roi_layer("tissue"),
                        sg_roi_rect(-2, -2, 8, 8, label = "tissue"))
proj <- sg_roi_project(sg_roi_frame(c(20, 20)), lyr)
fixed <- sg_roi_fix_geometry(proj)
sg_roi_rois(fixed)$area_px
```

sg_roi_flip_y *Flip an ROI's y-axis*

Description

Convert between image orientation (top-left origin, y down) and GIS orientation (bottom-left origin, y up). Self-inverse.

Usage

```
sg_roi_flip_y(roi, height, call = rlang::caller_env())
```

Arguments

roi	An sg_roi.
height	Numeric image height in pixels at the ROI's reference level.
call	The calling environment, for error reporting.

Value

An sg_roi with the y-axis flipped.

See Also

Other geometry: sg_roi_buffer(), sg_roi_check_containment(), sg_roi_check_geometry(), sg_roi_clamp(), sg_roi_contains(), sg_roi_distance(), sg_roi_fix_geometry(), sg_roi_overlaps(), sg_roi_rescale(), sg_roi_ring(), sg_roi_rois_overlap(), sg_roi_setops, sg_roi_simplify(), sg_roi_snap(), sg_roi_transform()

Examples

```
sg_roi_flip_y(sg_roi_rect(0, 0, 10, 5, label = "a"), height = 100)
```

sg_roi_frame *Declare the coordinate frame for histology ROIs*

Description

A metadata-only frame; no image reader, spectral backend, or pixel data is loaded. Coordinates have top-left origin, x right, y down. Pyramid dimensions are supplied explicitly, permitting anisotropic and non-power-of-two levels. Calibration is the physical size of a level-zero pixel along x and y.

Usage

```
sg_roi_frame(
  dims,
  level_dims = list(dims),
  pixel_size = c(1, 1),
  pixel_unit = "px",
  source = ""
)
```

Arguments

dims	Integer-valued width and height at level zero.
level_dims	List of width/height vectors, starting with dims.
pixel_size	Positive finite x/y pixel sizes at level zero.
pixel_unit	Unit of the calibration, e.g. "um" or "px".
source	Image identifier or path, recorded without reading the file.

Value

A sg_roi_frame for sg_roi_project() and sg_roi_transform().

Examples

```
frame <- sg_roi_frame(c(100, 80), list(c(100, 80), c(30, 20)), c(0.5, 0.8), "um")
sg_roi_dims(frame, 1)
```

---

sg_roi_freehand *Create a freehand ROI*

Description

A freehand trace is stored as a closed polygon, optionally simplified.

Usage

```
sg_roi_freehand(
  coords,
  label,
  level = 0L,
  simplify = 0,
  ...,
  call = rlang::caller_env()
)
```

Arguments

coords	Two-column numeric matrix of vertices in drawing order.
label	Single string class label.
level	Integer pyramid level the coordinates refer to. Default 0.
simplify	Non-negative Douglas-Peucker tolerance (dTolerance) applied to the closed polygon. 0 (default) keeps every vertex.
...	Additional named attributes stored on the ROI. The reserved names author, source, and id set those fields; all others become user attributes.
call	The calling environment, for error reporting.

Value

An sg_roi with POLYGON geometry.

See Also

Other rois: sg_roi_area(), sg_roi_bbox(), sg_roi_centroid(), sg_roi_circle(), sg_roi_ellipse(), sg_roi_from_geojson(), sg_roi_from_sf(), sg_roi_point(), sg_roi_polygon(), sg_roi_rect()

Examples

```
pts <- cbind(c(0, 5, 10, 5), c(0, 2, 0, -2))
r <- sg_roi_freehand(pts, label = "trace")
```

---

sg_roi_from_geojson *Convert a GeoJSON feature to an ROI*

Description

Turn a single parsed GeoJSON feature (a list with geometry and optional properties) into an sg_roi. Useful for consuming interactive drawing output.

Usage

```
sg_roi_from_geojson(
  feature,
  label = NULL,
  level = 0L,
  source = "manual",
  call = rlang::caller_env()
)
```

Arguments

feature	A parsed GeoJSON feature: a list with a geometry element (type and coordinates) and optional properties.
label	Single string class label; taken from properties$label when NULL.
level	Integer pyramid level. Default 0.
source	Provenance source string. Default "manual".
call	The calling environment, for error reporting.

Value

An sg_roi.

See Also

Other io: sg_roi_read_geojson(), sg_roi_read_qupath(), sg_roi_write_geojson(), sg_roi_write_qupath()

Other rois: sg_roi_area(), sg_roi_bbox(), sg_roi_centroid(), sg_roi_circle(), sg_roi_ellipse(), sg_roi_freehand(), sg_roi_from_sf(), sg_roi_point(), sg_roi_polygon(), sg_roi_rect()

Examples

```
feat <- list(geometry = list(type = "Polygon",
  coordinates = list(list(c(0, 0), c(4, 0), c(4, 4), c(0, 4), c(0, 0)))))
sg_roi_from_geojson(feat, label = "region")
```

---

sg_roi_from_sf *Create an ROI from an existing sf geometry*

Description

Create an ROI from an existing sf geometry

Usage

```
sg_roi_from_sf(geometry, label, level = 0L, ..., call = rlang::caller_env())
```

Arguments

geometry	An sf, sfc, or sfg geometry in image pixel coordinates.
label	Single string class label.
level	Integer pyramid level the coordinates refer to. Default 0.
...	Additional named attributes stored on the ROI. The reserved names author, source, and id set those fields; all others become user attributes.
call	The calling environment, for error reporting.

Value

An sg_roi wrapping the supplied geometry.

See Also

Other rois: sg_roi_area(), sg_roi_bbox(), sg_roi_centroid(), sg_roi_circle(), sg_roi_ellipse(), sg_roi_freehand(), sg_roi_from_geojson(), sg_roi_point(), sg_roi_polygon(), sg_roi_rect()

Examples

```
g <- sf::st_point(c(3, 4))
r <- sg_roi_from_sf(g, label = "imported", source = "imported")
```

---

sg_roi_layer *Create a layer*

Description

Create a layer

Usage

```
sg_roi_layer(
  name,
  labels = character(),
  style = sg_roi_style(),
  ...,
  call = rlang::caller_env()
)
```

Arguments

name	Single string layer name (unique within a project).
labels	Character vector; the controlled label vocabulary for the layer.
style	An annot_style, from sg_roi_style().
...	Additional named metadata stored on the layer.
call	The calling environment, for error reporting.

Value

An sg_roi_layer with no ROIs and a resolved colour mapping.

See Also

sg_roi_style(), sg_roi_layer_add()

Other layers: sg_roi_layer_add(), sg_roi_layer_n(), sg_roi_layer_remove(), sg_roi_layer_rois(), sg_roi_style()

Examples

```
lyr <- sg_roi_layer("tumour", labels = c("tumour", "stroma"))
sg_roi_layer_n(lyr)
```

`sg_roi_layer_add` *Add an ROI to a layer*

Description

Add an ROI to a layer

Usage

```
sg_roi_layer_add(layer, roi, call = rlang::caller_env())
```

Arguments

`layer`	An sg_roi_layer.
`roi`	An sg_roi.
`call`	The calling environment, for error reporting.

Value

A new sg_roi_layer with the ROI appended. The label vocabulary and colour mapping are extended if the ROI introduces a new label. A colliding ROI identifier is re-minted to preserve uniqueness. The input is unchanged.

See Also

Other layers: `sg_roi_layer()`, `sg_roi_layer_n()`, `sg_roi_layer_remove()`, `sg_roi_layer_rois()`, `sg_roi_style()`

Examples

```
lyr <- sg_roi_layer("tumour", labels = "tumour")
lyr <- sg_roi_layer_add(lyr, sg_roi_rect(0, 0, 10, 10, label = "tumour"))
sg_roi_layer_n(lyr)
```

sg_roi_layer_n *Number of ROIs in a layer*

Description

Number of ROIs in a layer

Usage

```
sg_roi_layer_n(layer, call = rlang::caller_env())
```

Arguments

layer	An sg_roi_layer.
call	The calling environment, for error reporting.

Value

An integer scalar.

See Also

Other layers: sg_roi_layer(), sg_roi_layer_add(), sg_roi_layer_remove(), sg_roi_layer_rois(), sg_roi_style()

Examples

```
lyr <- sg_roi_layer_add(sg_roi_layer("tissue"),
                        sg_roi_rect(0, 0, 4, 4, label = "tissue"))
sg_roi_layer_n(lyr)
```

sg_roi_layer_remove *Remove an ROI from a layer*

Description

Remove an ROI from a layer

Usage

```
sg_roi_layer_remove(layer, id, call = rlang::caller_env())
```

Arguments

layer	An sg_roi_layer.
id	Single string ROI identifier.
call	The calling environment, for error reporting.

Value

A new sg_roi_layer without the identified ROI. Warns if the ROI is not present and returns the layer unchanged.

See Also

Other layers: `sg_roi_layer()`, `sg_roi_layer_add()`, `sg_roi_layer_n()`, `sg_roi_layer_rois()`, `sg_roi_style()`

Examples

```
lyr <- sg_roi_layer_add(sg_roi_layer("tissue"),
                        sg_roi_rect(0, 0, 4, 4, label = "tissue", id = "r1"))
sg_roi_layer_n(sg_roi_layer_remove(lyr, "r1"))
```

---

sg_roi_layer_rois *Layer ROI table*

Description

Layer ROI table

Usage

```
sg_roi_layer_rois(layer, image = NULL, call = rlang::caller_env())
```

Arguments

layer	An sg_roi_layer.
image	Optional sg_roi_frame for accurate level-0 coordinate transforms.
call	The calling environment, for error reporting.

Value

An ROI table (see `sg_roi_rois()` for the column contract). A 0-row table with the same columns when the layer has no ROIs.

See Also

Other layers: `sg_roi_layer()`, `sg_roi_layer_add()`, `sg_roi_layer_n()`, `sg_roi_layer_remove()`, `sg_roi_style()`

Examples

```
lyr <- sg_roi_layer_add(sg_roi_layer("tissue"),
                        sg_roi_rect(0, 0, 4, 4, label = "tumour"))
lyr <- sg_roi_layer_add(lyr, sg_roi_rect(5, 5, 8, 8, label = "stroma"))
sg_roi_layer_rois(lyr)[, c("label", "area_px")]
```

sg_roi_layers *Layer summary table*

Description

Layer summary table

Usage

```
sg_roi_layers(project, call = rlang::caller_env())
```

Arguments

project	An sg_roi_project.
call	The calling environment, for error reporting.

Value

A tibble::tibble with one row per layer and the columns name (character), n_rois (integer), labels (list of character), visible (logical), locked (logical), and z (integer). A 0-row tibble with these columns when the project has no layers.

See Also

Other projects: sg_roi_add_layer(), sg_roi_add_roi(), sg_roi_project(), sg_roi_remove_layer(), sg_roi_remove_roi(), sg_roi_rois(), sg_roi_summary(), sg_roi_validate()

Examples

```
proj <- sg_roi_project(sg_roi_frame(c(20, 20)),
                       list(sg_roi_layer("tumour"), sg_roi_layer("stroma")))
sg_roi_layers(proj)
```

sg_roi_mask *Rasterise ROIs to a mask*

Description

Produce a binary, labelled, or multi-class integer mask from annotations. This is the package's core output artefact.

Usage

```
sg_roi_mask(
  x,
  type = c("binary", "labelled", "multiclass"),
  layer = NULL,
  label = NULL,
  level = 0L,
  background = 0L,
  touches = FALSE,
  dims = NULL,
  values = NULL,
  overlap = c("last", "first", "max", "min", "error", "bitor"),
  engine = c("stars", "terra"),
  call = rlang::caller_env()
)
```

Arguments

x	An sg_roi, sg_roi_layer, or sg_roi_project.
type	"binary" (a logical matrix), "labelled" (one integer id per ROI), or "multiclass" (one id per label class).
layer	Optional layer name(s) to restrict to (projects only).
label	Optional label(s) to restrict to.
level	Integer pyramid level at which to rasterise. Default 0.
background	Integer background value. Default 0.
touches	Logical; see the coverage contract. Default FALSE.
dims	Optional c(width, height); taken from the image (projects) or the geometry bounding box otherwise.
values	Optional named integer vector mapping labels to explicit class codes (e.g. c(specular = 1L, blood = 2L, shadow = 4L)), used only for type = "multiclass". When NULL (default) codes follow first-seen label order. Supply power-of-two codes together with overlap = "bitor" to build a bitfield mask.
overlap	How overlapping ROIs resolve: "last" (later z-order wins, the default), "first", "max", "min", "error" (abort on any overlap), or "bitor" (bitwise-OR the overlapping values, for bitfield masks; use with power-of-two values and background = 0).
engine	"stars" (reference) or "terra" (accelerated, if installed); the two produce identical masks.
call	The calling environment, for error reporting.

Details

Pixel-coverage contract. Pixel (i, j) covers the half-open square [j-1, j) x [i-1, i) with (0, 0) at the top-left corner of the top-left pixel; its centre is (j - 0.5, i - 0.5). With touches = FALSE (default) a pixel is included when its centre lies inside the polygon; with touches = TRUE a pixel is included if the polygon touches it at all. ROI coordinates use the image convention (top-left origin, y down).

Value

An sg_roi_mask: a matrix (logical for "binary", otherwise integer) with attributes legend (a tibble with columns value, label, layer, roi_id, n_px, colour), level, dims, type, and created. An all-background mask of the correct dimensions is returned when there are no ROIs.

See Also

sg_roi_mask_legend(), sg_roi_mask_stats()

Other masks: sg_roi_mask_boundary(), sg_roi_mask_derive(), sg_roi_mask_legend(), sg_roi_mask_preview(), sg_roi_mask_stack(), sg_roi_mask_stats()

Examples

```
lyr <- sg_roi_layer_add(sg_roi_layer("tissue"),
                        sg_roi_rect(2, 2, 8, 8, label = "tissue"))
proj <- sg_roi_project(sg_roi_frame(c(20, 20)), lyr)
m <- sg_roi_mask(proj, type = "multiclass")
table(as.integer(m))
```

---

sg_roi_mask_boundary *Object boundaries of a mask*

Description

Object boundaries of a mask

Usage

```
sg_roi_mask_boundary(mask, width = 1L, call = rlang::caller_env())
```

Arguments

mask	An sg_roi_mask.
width	Integer boundary thickness in pixels. Default 1.
call	The calling environment, for error reporting.

Value

A logical matrix marking boundary pixels (where a 4-neighbour has a different value).

See Also

Other masks: sg_roi_mask(), sg_roi_mask_derive(), sg_roi_mask_legend(), sg_roi_mask_preview(), sg_roi_mask_stack(), sg_roi_mask_stats()

Examples

```
lyr <- sg_roi_layer_add(sg_roi_layer("tissue"),
                        sg_roi_rect(2, 2, 8, 8, label = "tissue"))
m <- sg_roi_mask(sg_roi_project(sg_roi_frame(c(10, 10)), lyr), "labelled")
sg_roi_mask_boundary(m) * 1L
```

`sg_roi_mask_derive` *Derive a training mask from layer masks*

Description

Combine rasterised layer masks into the derived training mask `state WHERE anatomy == keep_label AND artefact == 0 A`
This is the cross-layer operation the four-layer HSI scheme relies on: it keeps a state (class) label only where the anatomy layer marks the region of interest, no artefact bit is set, and the state is actually labelled.

Usage

```
sg_roi_mask_derive(
  state,
  anatomy,
  artefact = NULL,
  keep_label = "wound",
  background = 0L,
  call = rlang::caller_env()
)
```

Arguments

`state`	An `sg_roi_mask` of state (class) codes.
`anatomy`	An `sg_roi_mask` of anatomy codes, sharing `state`'s dimensions and level.
`artefact`	Optional `sg_roi_mask` (bitfield); pixels with any artefact bit set are dropped. `NULL` (default) applies no artefact exclusion.
`keep_label`	The anatomy label whose region is kept (e.g. `"wound"`).
`background`	Integer background/unlabelled value. Default `0`.
`call`	The calling environment, for error reporting.

Value

An `sg_roi_mask` (multiclass) of the surviving state codes, with the state legend restricted to those classes and pixel counts recomputed.

See Also

`sg_roi_mask()`, `sg_roi_mask_stack()`

Other masks: `sg_roi_mask()`, `sg_roi_mask_boundary()`, `sg_roi_mask_legend()`, `sg_roi_mask_preview()`, `sg_roi_mask_stack()`, `sg_roi_mask_stats()`

Examples

```
lyr <- sg_roi_layer_add(sg_roi_layer("tissue"),
                        sg_roi_rect(2, 2, 8, 8, label = "tissue"))
proj <- sg_roi_project(sg_roi_frame(c(20, 20)), lyr)
# A single-layer project derives trivially; the four-layer HSI workflow passes
# separate anatomy/state/artefact masks. See the masks vignette.
m <- sg_roi_mask(proj, "multiclass")
sg_roi_mask_derive(m, m, keep_label = sg_roi_mask_legend(m)$label[1])
```

---

sg_roi_mask_legend *Legend of a mask*

Description

Legend of a mask

Usage

```
sg_roi_mask_legend(mask, call = rlang::caller_env())
```

Arguments

mask	An sg_roi_mask.
call	The calling environment, for error reporting.

Value

The mask's legend tibble::tibble (value, label, layer, roi_id, n_px, colour).

See Also

Other masks: sg_roi_mask(), sg_roi_mask_boundary(), sg_roi_mask_derive(), sg_roi_mask_preview(), sg_roi_mask_stack(), sg_roi_mask_stats()

Examples

```
lyr <- sg_roi_layer_add(sg_roi_layer("tissue"),
                        sg_roi_rect(2, 2, 8, 8, label = "tissue"))
proj <- sg_roi_project(sg_roi_frame(c(20, 20)), lyr)
sg_roi_mask_legend(sg_roi_mask(proj, "labelled"))
```

sg_roi_mask_preview *Downsampled preview of a mask*

Description

Nearest-neighbour downsample only (interpolating a label mask would invent labels that do not exist).

Usage

```
sg_roi_mask_preview(mask, max_dim = 1024L, call = rlang::caller_env())
```

Arguments

mask	An sg_roi_mask.
max_dim	Integer maximum dimension of the preview. Default 1024.
call	The calling environment, for error reporting.

Value

A downsampled sg_roi_mask with the legend pixel counts recomputed.

See Also

Other masks: sg_roi_mask(), sg_roi_mask_boundary(), sg_roi_mask_derive(), sg_roi_mask_legend(), sg_roi_mask_stack(), sg_roi_mask_stats()

Examples

```
lyr <- sg_roi_layer_add(sg_roi_layer("tissue"),
                        sg_roi_rect(20, 20, 80, 80, label = "tissue"))
m <- sg_roi_mask(sg_roi_project(sg_roi_frame(c(100, 100)), lyr), "labelled")
dim(sg_roi_mask_preview(m, max_dim = 25))
```

sg_roi_mask_stack *Stack per-layer masks into a 3D array*

Description

Stack per-layer masks into a 3D array

Usage

```
sg_roi_mask_stack(
  project,
  layers = NULL,
  level = 0L,
  call = rlang::caller_env()
)
```

Arguments

project	An sg_roi_project.
layers	Optional layer name(s) to include; all by default.
level	Integer pyramid level. Default 0.
call	The calling environment, for error reporting.

Value

A 3D integer array [y, x, layer], one multi-class plane per layer; the third dimension is named by layer.

See Also

Other masks: sg_roi_mask(), sg_roi_mask_boundary(), sg_roi_mask_derive(), sg_roi_mask_legend(), sg_roi_mask_preview(), sg_roi_mask_stats()

Examples

```
tumour <- sg_roi_layer_add(sg_roi_layer("tumour"),
                           sg_roi_rect(2, 2, 8, 8, label = "tumour"))
stroma <- sg_roi_layer_add(sg_roi_layer("stroma"),
                           sg_roi_rect(10, 10, 18, 18, label = "stroma"))
proj <- sg_roi_project(sg_roi_frame(c(20, 20)), list(tumour, stroma))
dim(sg_roi_mask_stack(proj))
```

---

sg_roi_mask_stats *Per-value statistics of a mask*

Description

Per-value statistics of a mask

Usage

```
sg_roi_mask_stats(mask, pixel_size = NULL, call = rlang::caller_env())
```

Arguments

mask	An sg_roi_mask.
pixel_size	Optional numeric c(x, y) physical pixel size for physical area at the mask's level (scale level-zero calibration before use on a coarser mask). When NULL, area_physical is NA.
call	The calling environment, for error reporting.

Value

A tibble::tibble with one row per mask value: value, label, n_px, area_px, area_physical, frac_total, bounding box (bbox_xmin, bbox_ymin, bbox_xmax, bbox_ymax), and centroid (centroid_x, centroid_y). A 0-row tibble when the mask is empty.

See Also

Other masks: sg_roi_mask(), sg_roi_mask_boundary(), sg_roi_mask_derive(), sg_roi_mask_legend(), sg_roi_mask_preview(), sg_roi_mask_stack()

Examples

```
lyr <- sg_roi_layer_add(sg_roi_layer("tissue"),
                        sg_roi_rect(2, 2, 8, 8, label = "tissue"))
proj <- sg_roi_project(sg_roi_frame(c(20, 20)), lyr)
sg_roi_mask_stats(sg_roi_mask(proj, "labelled"))
```

---

sg_roi_overlaps *Test whether two ROIs intersect*

Description

Test whether two ROIs intersect

Usage

```
sg_roi_overlaps(roi1, roi2, call = rlang::caller_env())
```

Arguments

roi1, roi2	sg_roi objects.
call	The calling environment, for error reporting.

Value

A logical scalar.

See Also

Other geometry: sg_roi_buffer(), sg_roi_check_containment(), sg_roi_check_geometry(), sg_roi_clamp(), sg_roi_contains(), sg_roi_distance(), sg_roi_fix_geometry(), sg_roi_flip_y(), sg_roi_rescale(), sg_roi_ring(), sg_roi_rois_overlap(), sg_roi_setops, sg_roi_simplify(), sg_roi_snap(), sg_roi_transform()

Examples

```
a <- sg_roi_rect(0, 0, 4, 4, label = "a")
sg_roi_overlaps(a, sg_roi_rect(2, 2, 6, 6, label = "b"))
sg_roi_overlaps(a, sg_roi_rect(10, 0, 14, 4, label = "c"))
```

---

sg_roi_point *Create a point ROI*

Description

Create a point ROI

Usage

```
sg_roi_point(x, y, label, level = 0L, ..., call = rlang::caller_env())
```

Arguments

x, y	Numeric coordinates of the point, in image pixels.
label	Single string class label.
level	Integer pyramid level the coordinates refer to. Default 0.
...	Additional named attributes stored on the ROI. The reserved names author, source, and id set those fields; all others become user attributes.
call	The calling environment, for error reporting.

Value

An sg_roi with POINT geometry.

See Also

sg_roi_rect(), sg_roi_polygon()

Other rois: sg_roi_area(), sg_roi_bbox(), sg_roi_centroid(), sg_roi_circle(), sg_roi_ellipse(), sg_roi_freehand(), sg_roi_from_geojson(), sg_roi_from_sf(), sg_roi_polygon(), sg_roi_rect()

Examples

```
r <- sg_roi_point(10, 20, label = "cell")
sg_roi_centroid(r)
```

`sg_roi_polygon` *Create a polygon ROI*

Description

Create a polygon ROI

Usage

```
sg_roi_polygon(
  coords,
  label,
  level = 0L,
  hole = NULL,
  ...,
  call = rlang::caller_env()
)
```

Arguments

`coords`	Either a two-column numeric matrix of exterior-ring vertices, or a list of such matrices where the first is the exterior ring and any others are holes. Rings are closed automatically.
`label`	Single string class label.
`level`	Integer pyramid level the coordinates refer to. Default `0`.
`hole`	Optional list of two-column matrices giving additional holes, combined with any holes supplied via `coords`.
`...`	Additional named attributes stored on the ROI. The reserved names `author`, `source`, and `id` set those fields; all others become user attributes.
`call`	The calling environment, for error reporting.

Value

An sg_roi with `POLYGON` geometry.

See Also

Other rois: `sg_roi_area()`, `sg_roi_bbox()`, `sg_roi_centroid()`, `sg_roi_circle()`, `sg_roi_ellipse()`, `sg_roi_freehand()`, `sg_roi_from_geojson()`, `sg_roi_from_sf()`, `sg_roi_point()`, `sg_roi_rect()`

Examples

```
sq <- matrix(c(0, 0, 10, 0, 10, 10, 0, 10), ncol = 2, byrow = TRUE)
r <- sg_roi_polygon(sq, label = "region")
```

sg_roi_project *Create an annotation project*

Description

Create an annotation project

Usage

```
sg_roi_project(image, layers = list(), ..., call = rlang::caller_env())
```

Arguments

image	An sg_roi_frame the annotations refer to.
layers	An sg_roi_layer, a list of them, or an empty list (default).
...	Additional named project metadata (e.g. name, description).
call	The calling environment, for error reporting.

Value

An sg_roi_project bundling the image, layers, metadata, and provenance (creation time, segmantR version, R version, author, and an edit log).

See Also

sg_roi_add_layer(), sg_roi_add_roi(), sg_roi_rois(), sg_roi_mask()

Other projects: sg_roi_add_layer(), sg_roi_add_roi(), sg_roi_layers(), sg_roi_remove_layer(), sg_roi_remove_roi(), sg_roi_rois(), sg_roi_summary(), sg_roi_validate()

Examples

```
sg_roi_project(sg_roi_frame(c(100, 100)), sg_roi_layer("tissue"))
```

sg_roi_read_geojson *Read ROIs from a GeoJSON file*

Description

Read ROIs from a GeoJSON file

Usage

```
sg_roi_read_geojson(
  path,
  layer_name = NULL,
  level = 0L,
  flip_y = FALSE,
  height = NULL,
  call = rlang::caller_env()
)
```

Arguments

path	Path to a GeoJSON file.
layer_name	Layer name for the imported ROIs; taken from the features' layer property when NULL.
level	Integer level to record when a feature has no level property.
flip_y	Logical; invert the y-axis on read. Default FALSE.
height	Image height in pixels, required when flip_y = TRUE.
call	The calling environment, for error reporting.

Value

A named list of sg_roi_layer objects (one per distinct layer property), or a single sg_roi_layer when layer_name is supplied.

See Also

Other io: sg_roi_from_geojson(), sg_roi_read_qupath(), sg_roi_write_geojson(), sg_roi_write_qupath()

Examples

```
lyr <- sg_roi_layer_add(sg_roi_layer("tissue"),
                        sg_roi_rect(2, 2, 8, 8, label = "tissue"))
p <- tempfile(fileext = ".geojson")
sg_roi_write_geojson(sg_roi_project(sg_roi_frame(c(20, 20)), lyr), p)
layers <- sg_roi_read_geojson(p)
sg_roi_layer_rois(layers$tissue)[, c("label", "area_px")]
```

---

sg_roi_read_qupath *Read ROIs from a QuPath GeoJSON file*

Description

Reads QuPath 0.4+ exports (a FeatureCollection or a bare feature array) and the legacy annotatR 0.1 dialect. `classification.name` (or the joined `names` of a derived class, e.g. `"Tumor: Positive"`) becomes the ROI label, the classification colour the layer style colour, and features without a classification are labelled `"unclassified"`. Everything else QuPath carries is kept, never invented, under each ROI's `attributes$qupath`: the object id and type, name, measurements (with explicit `value_state`), the image plane (zero-based `c`, `z`, `t`; `c = NA` for all channels), metadata, a nucleus geometry, and `roi_native = "ellipse"` with `geometry_fidelity = "approximated"` for polygonised ellipses. An segmantR id stored in `metadata.annotatr_roi_id` is restored as the ROI id.

Usage

```
sg_roi_read_qupath(
  path,
  layer_name = "qupath",
  level = 0L,
  call = rlang::caller_env()
)
```

Arguments

`path`	Path to a QuPath GeoJSON file.
`layer_name`	Layer name for the imported ROIs. Default `"qupath"`.
`level`	Integer level to record on the ROIs. Default `0`.
`call`	The calling environment, for error reporting.

Value

An sg_roi_layer.

See Also

Other io: `sg_roi_from_geojson()`, `sg_roi_read_geojson()`, `sg_roi_write_geojson()`, `sg_roi_write_qupath()`

Examples

```
lyr <- sg_roi_layer_add(sg_roi_layer("tissue"),
                        sg_roi_rect(2, 2, 8, 8, label = "tissue"))
p <- tempfile(fileext = ".geojson")
sg_roi_write_qupath(sg_roi_project(sg_roi_frame(c(20, 20)), lyr), p)
sg_roi_layer_n(sg_roi_read_qupath(p))
```

`sg_roi_rect` *Create a rectangular ROI*

Description

Create a rectangular ROI

Usage

```
sg_roi_rect(
  xmin,
  ymin,
  xmax,
  ymax,
  label,
  level = 0L,
  ...,
  call = rlang::caller_env()
)
```

Arguments

`xmin`, `ymin`, `xmax`, `ymax`	Numeric bounds in image pixels. `xmax`/`ymax` must exceed `xmin`/`ymin`.
`label`	Single string class label.
`level`	Integer pyramid level the coordinates refer to. Default `0`.
`...`	Additional named attributes stored on the ROI. The reserved names `author`, `source`, and `id` set those fields; all others become user attributes.
`call`	The calling environment, for error reporting.

Value

An sg_roi with `POLYGON` geometry (an axis-aligned rectangle).

See Also

Other rois: `sg_roi_area()`, `sg_roi_bbox()`, `sg_roi_centroid()`, `sg_roi_circle()`, `sg_roi_ellipse()`, `sg_roi_freehand()`, `sg_roi_from_geojson()`, `sg_roi_from_sf()`, `sg_roi_point()`, `sg_roi_polygon()`

Examples

```
r <- sg_roi_rect(0, 0, 10, 5, label = "tissue")
sg_roi_bbox(r)
```

sg_roi_remove_layer *Remove a layer from a project*

Description

Remove a layer from a project

Usage

```
sg_roi_remove_layer(project, name, call = rlang::caller_env())
```

Arguments

project	An sg_roi_project.
name	Single string layer name.
call	The calling environment, for error reporting.

Value

A new sg_roi_project without the named layer. The input is unchanged.

See Also

Other projects: sg_roi_add_layer(), sg_roi_add_roi(), sg_roi_layers(), sg_roi_project(), sg_roi_remove_roi(), sg_roi_rois(), sg_roi_summary(), sg_roi_validate()

Examples

```
proj <- sg_roi_project(sg_roi_frame(c(20, 20)),
                       list(sg_roi_layer("tumour"), sg_roi_layer("stroma")))
names(sg_roi_remove_layer(proj, "stroma")$layers)
```

sg_roi_remove_roi *Remove an ROI from a project*

Description

Remove an ROI from a project

Usage

```
sg_roi_remove_roi(project, id, call = rlang::caller_env())
```

Arguments

`project`	An sg_roi_project.
`id`	Single string ROI identifier.
`call`	The calling environment, for error reporting.

Value

A new sg_roi_project without the identified ROI. Warns and returns the project unchanged if the identifier is not found.

See Also

Other projects: `sg_roi_add_layer()`, `sg_roi_add_roi()`, `sg_roi_layers()`, `sg_roi_project()`, `sg_roi_remove_layer()`, `sg_roi_rois()`, `sg_roi_summary()`, `sg_roi_validate()`

Examples

```
lyr <- sg_roi_layer_add(sg_roi_layer("tissue"),
                        sg_roi_rect(2, 2, 8, 8, label = "tissue", id = "r1"))
proj <- sg_roi_project(sg_roi_frame(c(20, 20)), lyr)
nrow(sg_roi_rois(sg_roi_remove_roi(proj, "r1")))
```

`sg_roi_rescale` *Rescale an ROI between pyramid levels*

Description

Transform an ROI's coordinates from one pyramid level to another, assuming a power-of-two pyramid. For an exact transform against a real image pyramid (which need not be power-of-two), use `sg_roi_transform()` with the sg_roi_frame.

Usage

```
sg_roi_rescale(roi, from_level, to_level, call = rlang::caller_env())
```

Arguments

`roi`	An sg_roi.
`from_level`, `to_level`	Integer source and target pyramid levels.
`call`	The calling environment, for error reporting.

Value

An sg_roi with coordinates at `to_level` and its `level` field set to `to_level`. The transform is exactly invertible.

See Also

sg_roi_transform()

Other geometry: sg_roi_buffer(), sg_roi_check_containment(), sg_roi_check_geometry(), sg_roi_clamp(), sg_roi_contains(), sg_roi_distance(), sg_roi_fix_geometry(), sg_roi_flip_y(), sg_roi_overlaps(), sg_roi_ring(), sg_roi_rois_overlap(), sg_roi_setops, sg_roi_simplify(), sg_roi_snap(), sg_roi_transform()

Examples

```
r <- sg_roi_rect(0, 0, 100, 100, label = "a")
r2 <- sg_roi_rescale(r, from_level = 0, to_level = 2)
identical(
  sf::st_coordinates(sf::st_geometry(r)),
  sf::st_coordinates(sf::st_geometry(sg_roi_rescale(r2, 2, 0)))
)
```

---

sg_roi_ring *Ring (annulus) along an ROI margin*

Description

Build a band straddling an ROI's boundary – the natural shape for a "penumbra" region along an injury or wound margin. The ring extends `outer` units outside the boundary and `inner` units inside it, formed as the set difference of the outward and inward buffers.

Usage

```
sg_roi_ring(
  roi,
  outer,
  inner = 0,
  label = roi$label,
  call = rlang::caller_env()
)
```

Arguments

`roi`	An sg_roi.
`outer`	Non-negative outward width (dilation) of the ring.
`inner`	Non-negative inward width (erosion) of the ring. Default `0`, a purely external band.
`label`	Label for the derived ring ROI. Default the source ROI's label.
`call`	The calling environment, for error reporting.

Value

A derived sg_roi (an annulus) with `source = "derived"`.

See Also

sg_roi_buffer(), sg_roi_difference()

Other geometry: sg_roi_buffer(), sg_roi_check_containment(), sg_roi_check_geometry(), sg_roi_clamp(), sg_roi_contains(), sg_roi_distance(), sg_roi_fix_geometry(), sg_roi_flip_y(), sg_roi_overlaps(), sg_roi_rescale(), sg_roi_rois_overlap(), sg_roi_setops, sg_roi_simplify(), sg_roi_snap(), sg_roi_transform()

Examples

```
inj <- sg_roi_rect(10, 10, 20, 20, label = "injury")
sg_roi_ring(inj, outer = 3, label = "penumbra")
```

---

sg_roi_rois *Query the ROIs of a project*

Description

Return a tidy table of ROIs, optionally filtered by layer and/or label.

Usage

```
sg_roi_rois(project, layer = NULL, label = NULL, call = rlang::caller_env())
```

Arguments

project	An sg_roi_project.
layer	Optional single layer name to filter to.
label	Optional character vector of labels to filter to.
call	The calling environment, for error reporting.

Details

Coordinates in the geometry column, together with area_px and the centroid, are expressed at pyramid level 0; the level column records the level at which each ROI was originally defined.

Value

A tibble::tibble with one row per ROI and, in this exact order, the columns:

roi_id Unique identifier (character).

layer Layer name (character).

label Class label (character).

geom_type Geometry type, e.g. "POLYGON" (character).

level Reference pyramid level (integer).

area_px Area in level-0 pixels (double); NA for points/lines.

`centroid_x`,`centroid_y` Centroid at level 0 (double).

`n_vertices` Vertex count (integer).

`created`,`modified` Timestamps (POSIXct).

`author` Author (character).

`source` Provenance source (character).

`geometry` The `sf` geometry column (`sfc`), at level 0.

Returns a 0-row tibble with these columns if no ROIs match.

See Also

`sg_roi_layers()`, `sg_roi_mask()`

Other projects: `sg_roi_add_layer()`, `sg_roi_add_roi()`, `sg_roi_layers()`, `sg_roi_project()`, `sg_roi_remove_layer()`, `sg_roi_remove_roi()`, `sg_roi_summary()`, `sg_roi_validate()`

Examples

```
lyr <- sg_roi_layer_add(sg_roi_layer("tissue"),
                        sg_roi_rect(2, 2, 8, 8, label = "tissue"))
proj <- sg_roi_project(sg_roi_frame(c(20, 20)), lyr)
sg_roi_rois(proj)
```

---

sg_roi_rois_overlap *Pairwise ROI overlaps within a layer*

Description

Pairwise ROI overlaps within a layer

Usage

```
sg_roi_rois_overlap(layer, call = rlang::caller_env())
```

Arguments

`layer`	An sg_roi_layer.
`call`	The calling environment, for error reporting.

Value

A tibble::tibble with columns `id_a`, `id_b` (character), `overlap_px` (double, level-0 pixels), and `jaccard` (double). A 0-row tibble with these columns when no ROIs overlap.

See Also

Other geometry: sg_roi_buffer(), sg_roi_check_containment(), sg_roi_check_geometry(), sg_roi_clamp(), sg_roi_contains(), sg_roi_distance(), sg_roi_fix_geometry(), sg_roi_flip_y(), sg_roi_overlaps(), sg_roi_rescale(), sg_roi_ring(), sg_roi_setops, sg_roi_simplify(), sg_roi_snap(), sg_roi_transform()

Examples

```
lyr <- sg_roi_layer_add(sg_roi_layer("cells"),
                        sg_roi_rect(0, 0, 4, 4, label = "cell"))
lyr <- sg_roi_layer_add(lyr, sg_roi_rect(2, 2, 6, 6, label = "cell"))
sg_roi_rois_overlap(lyr)
```

---

sg_roi_setops *Set operations on ROIs*

Description

Combine ROIs geometrically. Each accepts several sg_roi via ..., or a single sg_roi_layer. Multi-part results are returned as one ROI with MULTIPOLYGON geometry (the return type is always a single sg_roi).

Usage

```
sg_roi_union(..., call = rlang::caller_env())

sg_roi_intersect(..., call = rlang::caller_env())

sg_roi_difference(..., call = rlang::caller_env())

sg_roi_symdiff(..., call = rlang::caller_env())
```

Arguments

...	Two or more sg_roi, or a single sg_roi_layer.
call	The calling environment, for error reporting.

Value

A single sg_roi with source = "derived".

See Also

Other geometry: sg_roi_buffer(), sg_roi_check_containment(), sg_roi_check_geometry(), sg_roi_clamp(), sg_roi_contains(), sg_roi_distance(), sg_roi_fix_geometry(), sg_roi_flip_y(), sg_roi_overlaps(), sg_roi_rescale(), sg_roi_ring(), sg_roi_rois_overlap(), sg_roi_simplify(), sg_roi_snap(), sg_roi_transform()

Examples

```
a <- sg_roi_rect(0, 0, 6, 6, label = "x")
b <- sg_roi_rect(4, 4, 10, 10, label = "x")
sg_roi_area(sg_roi_intersect(a, b))
```

`sg_roi_simplify` *Simplify an ROI*

Description

Simplify an ROI

Usage

```
sg_roi_simplify(roi, tolerance, call = rlang::caller_env())
```

Arguments

`roi`	An sg_roi.
`tolerance`	Non-negative Douglas-Peucker tolerance (`dTolerance`).
`call`	The calling environment, for error reporting.

Value

An sg_roi with simplified geometry.

See Also

Other geometry: `sg_roi_buffer()`, `sg_roi_check_containment()`, `sg_roi_check_geometry()`, `sg_roi_clamp()`, `sg_roi_contains()`, `sg_roi_distance()`, `sg_roi_fix_geometry()`, `sg_roi_flip_y()`, `sg_roi_overlaps()`, `sg_roi_rescale()`, `sg_roi_ring()`, `sg_roi_rois_overlap()`, `sg_roi_setops`, `sg_roi_snap()`, `sg_roi_transform()`

Examples

```
sg_roi_simplify(sg_roi_circle(50, 50, 10, n_seg = 128, label = "a"), 1)
```

sg_roi_snap *Snap ROI vertices to a grid*

Description

Snap ROI vertices to a grid

Usage

```
sg_roi_snap(roi, grid = 1, call = rlang::caller_env())
```

Arguments

roi	An sg_roi.
grid	Positive numeric grid spacing in pixels. Default 1.
call	The calling environment, for error reporting.

Value

An sg_roi with vertices rounded to the nearest grid multiple.

See Also

Other geometry: sg_roi_buffer(), sg_roi_check_containment(), sg_roi_check_geometry(), sg_roi_clamp(), sg_roi_contains(), sg_roi_distance(), sg_roi_fix_geometry(), sg_roi_flip_y(), sg_roi_overlaps(), sg_roi_rescale(), sg_roi_ring(), sg_roi_rois_overlap(), sg_roi_setops, sg_roi_simplify(), sg_roi_transform()

Examples

```
sg_roi_snap(sg_roi_point(3.4, 5.6, label = "a"), grid = 1)
```

sg_roi_style *Layer style*

Description

Construct a style specification for an sg_roi_layer.

Usage

```
sg_roi_style(
  colour = NULL,
  fill_alpha = 0.3,
  stroke_width = 2,
  visible = TRUE,
  locked = FALSE,
  z = 1L,
  call = rlang::caller_env()
)
```

Arguments

colour	Either NULL (default; the Okabe-Ito colourblind-safe palette is cycled over the layer's labels), a single colour recycled across labels, or a named character vector mapping labels to colours. The style is stored with the resolved mapping so colours are stable across sessions.
fill_alpha	Fill opacity in [0, 1]. Default 0.3.
stroke_width	Stroke width in pixels. Default 2.
visible	Logical; whether the layer is drawn. Default TRUE.
locked	Logical; whether the layer is protected from edits. Default FALSE.
z	Integer draw order (higher is drawn later, on top). Affects mask overlap resolution. Default 1.
call	The calling environment, for error reporting.

Value

A named list of class annot_style.

See Also

Other layers: sg_roi_layer(), sg_roi_layer_add(), sg_roi_layer_n(), sg_roi_layer_remove(), sg_roi_layer_rois()

Examples

```
sg_roi_style(fill_alpha = 0.5, z = 2)
```

sg_roi_summary *Summarise a project*

Description

Summarise a project

Usage

```
sg_roi_summary(project, call = rlang::caller_env())
```

Arguments

project	An sg_roi_project.
call	The calling environment, for error reporting.

Value

An sg_roi_summary object: a list with n_layers, n_rois, per-label ROI counts, and total area per label.

See Also

Other projects: sg_roi_add_layer(), sg_roi_add_roi(), sg_roi_layers(), sg_roi_project(), sg_roi_remove_layer(), sg_roi_remove_roi(), sg_roi_rois(), sg_roi_validate()

Examples

```
lyr <- sg_roi_layer_add(sg_roi_layer("tissue"),
                        sg_roi_rect(2, 2, 8, 8, label = "tissue"))
proj <- sg_roi_project(sg_roi_frame(c(20, 20)), lyr)
sg_roi_summary(proj)
```

sg_roi_transform *Transform an ROI between pyramid levels using real image dimensions*

Description

Unlike sg_roi_rescale() (which assumes a power-of-two pyramid), this uses the actual level dimensions of an sg_roi_frame, so it is exact for pyramids with any downsampling ratio.

Usage

```
sg_roi_transform(roi, from_level, to_level, img, call = rlang::caller_env())
```

Arguments

`roi`	An sg_roi.
`from_level, to_level`	Integer source and target pyramid levels.
`img`	The sg_roi_frame whose pyramid defines the level dimensions.
`call`	The calling environment, for error reporting.

Value

An sg_roi with coordinates at `to_level`. Exactly invertible.

See Also

`sg_roi_rescale()`

Other geometry: `sg_roi_buffer()`, `sg_roi_check_containment()`, `sg_roi_check_geometry()`, `sg_roi_clamp()`, `sg_roi_contains()`, `sg_roi_distance()`, `sg_roi_fix_geometry()`, `sg_roi_flip_y()`, `sg_roi_overlaps()`, `sg_roi_rescale()`, `sg_roi_ring()`, `sg_roi_rois_overlap()`, `sg_roi_setops`, `sg_roi_simplify()`, `sg_roi_snap()`

Examples

```
img <- sg_roi_frame(c(100, 100), list(c(100, 100), c(50, 50), c(25, 25)))
r <- sg_roi_rect(0, 0, 100, 100, label = "a")
sg_roi_transform(r, from_level = 0, to_level = 2, img = img)
```

---

sg_roi_validate *Validate a project's invariants*

Description

Check that layer names are unique, ROI identifiers are unique across the project, and all geometries are valid.

Usage

```
sg_roi_validate(project, call = rlang::caller_env())
```

Arguments

`project`	An sg_roi_project.
`call`	The calling environment, for error reporting.

Value

The project, invisibly, if valid; otherwise an error describing the violated invariant.

See Also

`sg_roi_check_geometry()`

Other projects: `sg_roi_add_layer()`, `sg_roi_add_roi()`, `sg_roi_layers()`, `sg_roi_project()`, `sg_roi_remove_layer()`, `sg_roi_remove_roi()`, `sg_roi_rois()`, `sg_roi_summary()`

Examples

```
lyr <- sg_roi_layer_add(sg_roi_layer("tissue"),
                        sg_roi_rect(2, 2, 8, 8, label = "tissue"))
proj <- sg_roi_project(sg_roi_frame(c(20, 20)), lyr)
sg_roi_validate(proj)
```

---

sg_roi_write_geojson *Write a project's ROIs to GeoJSON*

Description

Write a project's ROIs to GeoJSON

Usage

```
sg_roi_write_geojson(
  project,
  path,
  layer = NULL,
  level = 0L,
  flip_y = FALSE,
  overwrite = FALSE,
  call = rlang::caller_env()
)
```

Arguments

`project`	An sg_roi_project.
`path`	Output file path.
`layer`	Optional layer filter.
`level`	Integer pyramid level for the coordinates. Default `0`. It is written to each feature's `level` property, which `sg_roi_read_geojson()` restores.
`flip_y`	Logical; invert the y-axis (for GIS-oriented consumers). Default `FALSE` (image orientation preserved). The consequence is documented in the interop vignette.
`overwrite`	Logical; overwrite an existing file. Default `FALSE`.
`call`	The calling environment, for error reporting.

Value

The output path, invisibly.

See Also

sg_roi_read_geojson(), sg_roi_write_qupath()

Other io: sg_roi_from_geojson(), sg_roi_read_geojson(), sg_roi_read_qupath(), sg_roi_write_qupath()

Examples

```
lyr <- sg_roi_layer_add(sg_roi_layer("tissue"),
                        sg_roi_rect(2, 2, 8, 8, label = "tissue"))
proj <- sg_roi_project(sg_roi_frame(c(20, 20)), lyr)
p <- tempfile(fileext = ".geojson")
sg_roi_write_geojson(proj, p)
```

---

sg_roi_write_qupath *Write a project's ROIs as QuPath GeoJSON*

Description

The default `dialect = "qupath"` writes what QuPath 0.4 and later read natively (verified with QuPath 0.7.0): `objectType`, `classification` with an `[r, g, b]` colour, `isLocked` for locked ROIs, a deterministic UUID per ROI as the feature id (QuPath replaces non-UUID ids), and segmantR's own ROI id, layer, source and level in `properties.metadata`. `dialect = "legacy"` writes the annotatR 0.1 form (`object_type`, signed `colorRGB`), which QuPath still reads with a deprecation warning.

Usage

```
sg_roi_write_qupath(
  project,
  path,
  layer = NULL,
  level = 0L,
  overwrite = FALSE,
  dialect = c("qupath", "legacy"),
  call = rlang::caller_env()
)
```

Arguments

`project`	An sg_roi_project.
`path`	Output file path.
`layer`	Optional layer filter.
`level`	Integer pyramid level for the coordinates. Default `0`. It is recorded as `annotatr_level` in the feature metadata of the `"qupath"` dialect.
`overwrite`	Logical; overwrite an existing file. Default `FALSE`.
`dialect`	`"qupath"` (default) or `"legacy"`.
`call`	The calling environment, for error reporting.

Details

QuPath keeps one colour per classification name, so identical labels with different layer colours collapse to the first colour QuPath sees.

Value

The output path, invisibly.

See Also

sg_roi_read_qupath(), sg_roi_write_geojson()

Other io: sg_roi_from_geojson(), sg_roi_read_geojson(), sg_roi_read_qupath(), sg_roi_write_geojson()

Examples

```
lyr <- sg_roi_layer_add(sg_roi_layer("tissue"),
                        sg_roi_rect(2, 2, 8, 8, label = "tissue"))
proj <- sg_roi_project(sg_roi_frame(c(20, 20)), lyr)
p <- tempfile(fileext = ".geojson")
sg_roi_write_qupath(proj, p)
```

---

sg_run_app *Launch the segmantR Shiny application*

Description

Starts the interactive Shiny application for cell segmentation, annotation, and model training. The app is built by sg_app() from package code; inst/shiny/segmantR/app.R is a thin wrapper for shiny::runApp() on the installed directory.

Usage

```
sg_run_app(image = NULL, mask = NULL, port = NULL, launch.browser = TRUE)
```

Arguments

image	An sg_image object to pre-load, or NULL.
mask	An sg_mask object to pre-load, or NULL.
port	Integer port number, or NULL to use the default.
launch.browser	Logical, whether to open the app in a browser (default TRUE).

Details

For compatibility with code that starts the bundled app directory directly, the pre-loaded objects are also stored in the option segmantR.app_env while the app runs; the previous value is restored when it stops. sg_app() itself does not use options.

Value

NULL, invisibly.

Examples

```
if (interactive()) {
  sg_run_app(image = sg_example_image("fluorescence_nuclei"))
}
```

`sg_segment_cellpose` *Cellpose cell segmentation*

Description

Segments cells using the Cellpose deep learning model via `reticulate`. Requires a working Python installation with `cellpose` installed. Use `sg_setup_python()` to configure the environment.

Usage

```
sg_segment_cellpose(
  image,
  model = c("cyto3", "cyto2", "nuclei", "tissuenet", "livecell", "custom"),
  channels = list(cytoplasm = 0L, nucleus = 1L),
  diameter = NULL,
  flow_threshold = 0.4,
  cellprob_threshold = 0,
  custom_model_path = NULL,
  batch_size = 8L,
  tile = TRUE
)
```

Arguments

`image`	An `sg_image` object.
`model`	Character string specifying the Cellpose model. One of `"cyto3"` (default), `"cyto2"`, `"nuclei"`, `"tissuenet"`, `"livecell"`, or `"custom"`.
`channels`	A named list with elements `cytoplasm` and `nucleus`, each an integer channel index (0-based, following Cellpose convention). Default is `list(cytoplasm = 0L, nucleus = 1L)`.
`diameter`	Numeric or `NULL`. Estimated cell diameter in pixels. If `NULL`, Cellpose estimates the diameter automatically.
`flow_threshold`	Numeric; flow error threshold for mask filtering. Default is `0.4`.
`cellprob_threshold`	Numeric; cell probability threshold. Default is `0.0`.

custom_model_path
: Character string or NULL. Path to a custom-trained Cellpose model. Required when model = "custom".

batch_size
: Integer; number of images to process in parallel on the GPU. Default is 8L.

tile
: Logical; if TRUE (default), large images are processed in tiles to reduce memory usage.

Value

An sg_mask object with labelled cell regions.

Examples

```
pixels <- array(runif(300), dim = c(10, 10, 3))
img <- new_sg_image(pixels, channels = c("R", "G", "B"))
## Not run:
# Requires Python with Cellpose (see sg_setup_python())
mask <- sg_segment_cellpose(img, model = "cyto3")

## End(Not run)
```

---

sg_segment_mesmer *Mesmer (DeepCell) cell segmentation*

Description

Segments cells using the Mesmer deep learning model from the DeepCell library via reticulate. Mesmer is designed for multiplexed tissue imaging and can segment both nuclear and whole-cell compartments.

Usage

```
sg_segment_mesmer(
  image,
  compartment = c("whole-cell", "nuclear", "both"),
  image_mpp = NULL
)
```

Arguments

image
: An sg_image object. Must have at least two channels: a nuclear channel (first) and a membrane/cytoplasm channel (second).

compartment
: Character string specifying which compartment to segment. One of "whole-cell" (default), "nuclear", or "both".

image_mpp
: Numeric scalar or NULL. Microns per pixel of the input image. If NULL, the value is taken from the image resolution metadata; if that is also unavailable, 0.5 is used as a default.

Details

Requires a working Python installation with `deepcell` installed.

Value

An `sg_mask` object (for `"whole-cell"` or `"nuclear"`) or a named list of two `sg_mask` objects (for `"both"`).

Examples

```
pixels <- array(runif(200), dim = c(10, 10, 2))
img <- new_sg_image(pixels, channels = c("nuclear", "membrane"))
## Not run:
# Requires Python with DeepCell
mask <- sg_segment_mesmer(img, compartment = "whole-cell")

## End(Not run)
```

---

sg_segment_propagate *Voronoi propagation from nuclear seeds*

Description

Expands nuclear seed regions outward using Voronoi-style propagation, optionally constrained by a membrane signal. When available, `EBImage::propagate()` is used for higher performance; otherwise a pure-R fallback is used.

Usage

```
sg_segment_propagate(
  image,
  nuclear_mask,
  membrane_image = NULL,
  lambda = 0.01,
  expand_max = 20L,
  engine = c("auto", "voronoi_r", "ebimage")
)
```

Arguments

`image`	An `sg_image` object.
`nuclear_mask`	An `sg_mask` object providing the nuclear seed labels.
`membrane_image`	An `sg_image` object (single channel) used as a cost surface to penalise crossing membranes. If `NULL`, propagation proceeds uniformly.
`lambda`	Numeric weight for the membrane penalty in the cost function. Higher values make propagation stop more readily at membranes. Default is `0.01`.

expand_max	Integer; maximum number of propagation iterations. Default is 20L.
engine	"auto" (default, historic behaviour: EBImage when it is installed and no membrane image is given, otherwise pure R), "voronoi_r" (always the pure-R engine) or "ebimage" (always EBImage; errors instead of falling back).

Value

An sg_mask object with expanded cell labels.

Examples

```
set.seed(42)
pixels <- matrix(runif(400), nrow = 20, ncol = 20)
img <- new_sg_image(pixels)
seeds <- matrix(0L, nrow = 20, ncol = 20)
seeds[5, 5] <- 1L
seeds[15, 15] <- 2L
nuc_mask <- new_sg_mask(seeds)
result <- sg_segment_propagate(img, nuclear_mask = nuc_mask)
print(result)
```

---

sg_segment_stardist *StarDist cell segmentation*

Description

Segments cells (typically nuclei) using the StarDist deep learning model via reticulate. Requires a working Python installation with stardist and tensorflow installed. Use sg_setup_python() to configure the environment.

Usage

```
sg_segment_stardist(
  image,
 model = c("2D_versatile_fluo", "2D_versatile_he", "2D_paper_dsb2018", "custom"),
  channel = 1L,
  prob_thresh = 0.5,
  nms_thresh = 0.4,
  scale = NULL,
  custom_model_path = NULL,
  n_tiles = NULL
)
```

Arguments

image	An sg_image object.
model	Character string specifying the StarDist model. One of "2D_versatile_fluo" (default), "2D_versatile_he", "2D_paper_dsb2018", or "custom".

channel	Integer index of the image channel to segment. Default is 1L.
prob_thresh	Numeric; object probability threshold. Objects with probability below this value are discarded. Default is 0.5.
nms_thresh	Numeric; non-maximum suppression overlap threshold. Overlapping detections above this threshold are merged. Default is 0.4.
scale	Numeric vector of length 1 or 2, or NULL. Scaling factor applied to the image before prediction. If NULL, no scaling is applied.
custom_model_path	Character string or NULL. Path to a custom-trained StarDist model directory. Required when model = "custom".
n_tiles	Integer vector of length 2 or NULL. Number of tiles in each dimension for processing large images. If NULL, StarDist determines tiling automatically.

Value

An sg_mask object with labelled cell regions.

Examples

```
pixels <- matrix(runif(400), nrow = 20, ncol = 20)
img <- new_sg_image(pixels)
## Not run:
# Requires Python with StarDist (see sg_setup_python())
mask <- sg_segment_stardist(img, model = "2D_versatile_fluo")

## End(Not run)
```

---

sg_segment_threshold *Threshold-based cell segmentation*

Description

Segments cells in an sg_image by applying a global or local intensity threshold, followed by morphological cleanup and connected-component labelling with size filtering.

Usage

```
sg_segment_threshold(
  image,
  channel = 1L,
  method = c("otsu", "adaptive", "triangle"),
  block_size = 51L,
  offset = 0.05,
  morphology = list(open = 5L, fill_holes = TRUE),
  min_area = 50L,
  max_area = 5000L
)
```

Arguments

image	An sg_image object.
channel	Integer index of the image channel to threshold. Default is 1L.
method	Character string specifying the thresholding method. One of "otsu" (default), "adaptive", or "triangle".
block_size	Integer block size for adaptive thresholding. Must be a positive odd integer. Only used when method = "adaptive". Default is 51L.
offset	Numeric offset subtracted from the local mean in adaptive thresholding. Default is 0.05.
morphology	A named list controlling morphological cleanup: open Integer; structuring element size for opening (erosion then dilation). Set to 0L to skip. Default 5L. fill_holes Logical; whether to fill holes inside objects. Default TRUE.
min_area	Integer; minimum object area in pixels. Objects smaller than this are removed. Default is 50L.
max_area	Integer; maximum object area in pixels. Objects larger than this are removed. Default is 5000L.

Details

Pixels that are NA, such as nodata values of a raster, count as background.

Value

An sg_mask object with labelled cell regions.

Examples

```
set.seed(42)
pixels <- matrix(runif(400), nrow = 20, ncol = 20)
img <- new_sg_image(pixels)
mask <- sg_segment_threshold(img, method = "otsu")
print(mask)
```

---

sg_segment_watershed *Marker-controlled watershed segmentation*

Description

Segments cells using a marker-controlled watershed algorithm. Seeds (markers) are detected automatically via distance-transform peaks or h-minima inside the Otsu foreground of channel. The watershed then floods from these seeds, level by level along the surface, so that object boundaries follow its ridges; pixels outside the foreground are never flooded (only expand_pixels grows objects beyond it).

Usage

```
sg_segment_watershed(
  image,
  channel = 1L,
  seed_method = c("h_minima", "distance", "markers"),
  h = 0.05,
  min_distance = 10L,
  expand_method = c("voronoi", "dilation"),
  expand_pixels = 3L,
  membrane_channel = NULL
)
```

Arguments

image	An sg_image object.
channel	Integer index of the image channel used for segmentation. Default is 1L.
seed_method	Character string specifying how seeds are generated. One of "h_minima" (default), "distance", or "markers".
h	Numeric; height parameter for h-minima or distance threshold fraction for seed detection. Default is 0.05.
min_distance	Integer; minimum pixel distance between seeds. Default is 10L.
expand_method	Character string specifying how seed regions are expanded. One of "voronoi" (default) or "dilation".
expand_pixels	Integer; number of pixels to expand beyond the initial watershed regions. Default is 3L.
membrane_channel	Integer or NULL. If provided, this channel is used as the topographic surface for watershed flooding. If NULL, the inverted channel image is used.

Value

An sg_mask object with labelled cell regions.

Examples

```
set.seed(42)
pixels <- matrix(runif(400), nrow = 20, ncol = 20)
img <- new_sg_image(pixels)
mask <- sg_segment_watershed(img, seed_method = "distance", h = 0.3)
print(mask)
```

sg_select_channel *Select one analysis channel from an image*

Description

Produces a single-channel sg_image from a channel index, a channel or band name, a band wavelength, or one of the registered band operations band_mean, ratio and normalized_difference. No free expressions are accepted. The result records the source bands, wavelengths, operation, value semantics and a transform digest so declarative protocol runs and direct calls stay comparable.

Usage

```
sg_select_channel(
  image,
  channel = 1L,
  channel_name = NULL,
  wavelength_nm = NULL,
  wavelength_tolerance_nm = 5,
  band_operation = "none",
  band_a_nm = NULL,
  band_b_nm = NULL,
  band_min_nm = NULL,
  band_max_nm = NULL
)
```

Arguments

image	An sg_image object.
channel	1-based channel index (used when no other selector is given).
channel_name	Channel or band name.
wavelength_nm	Band centre wavelength to select.
wavelength_tolerance_nm	Maximum distance to the nearest band centre.
band_operation	One of "none", "band_mean", "ratio", "normalized_difference".
band_a_nm, band_b_nm	Wavelengths of bands a and b for ratio and normalized_difference.
band_min_nm, band_max_nm	Wavelength interval for band_mean.

Details

Band operations require band wavelengths (see new_sg_image() bands or sg_read_envi()) and refuse images whose value_semantics is "unknown" for ratio and normalized_difference, because such ratios are only interpretable for a declared value scale. Non-finite results (division by zero) are replaced by the smallest finite result and counted in provenance$band_selection$non_finite_replaced.

Value

A single-channel sg_image.

Examples

```
cube <- array(runif(5 * 5 * 3), dim = c(5, 5, 3))
img <- new_sg_image(cube, channels = c("b450", "b550", "b650"),
                    bands = data.frame(wavelength_nm = c(450, 550, 650)),
                    value_semantics = "reflectance")
sg_select_channel(img, wavelength_nm = 552)$channels
nd <- sg_select_channel(img, band_operation = "normalized_difference",
                        band_a_nm = 650, band_b_nm = 550)
nd$provenance$band_selection$operation
```

---

sg_setup_python *Set up a Python environment for deep learning backends*

Description

Creates and configures a Python virtual environment or conda environment with the requested deep learning segmentation backends (Cellpose, StarDist) installed.

Usage

```
sg_setup_python(
  envname = "segmantr",
  method = c("auto", "conda", "virtualenv"),
  backends = c("cellpose", "stardist"),
  gpu = TRUE
)
```

Arguments

envname	Character string naming the Python environment. Default is "segmantr".
method	Character string specifying the environment type. One of "auto" (default), "conda", or "virtualenv".
backends	Character vector of backends to install. Supported values are "cellpose" and "stardist".
gpu	Logical; if TRUE (default), installs the GPU-capable tensorflow build for StarDist instead of tensorflow-cpu. Cellpose is always installed as cellpose; it uses a GPU when the installed PyTorch build supports it.

Value

Invisibly returns TRUE on success.

Selecting Python

segmantR never starts, downloads or installs Python by itself; without a configured Python, recent versions of 'reticulate' would provision one with 'uv'. The deep learning backends count as available only when Python is already running in the session (for example after `reticulate::use_virtualenv("segmantr")` and `reticulate::py_config()`), when the environment variable `RETICULATE_PYTHON` or `RETICULATE_PYTHON_ENV` names a Python, or when the option `segmantR.python` is `TRUE`, which lets 'reticulate' find (or provision) a Python on first use.

Examples

```
## Not run:
sg_setup_python(backends = "cellpose")

## End(Not run)
```

---

sg_slide_display_ranges

Slide-wide display ranges from sampled image blocks

Description

Estimates fixed per-channel display ranges for a whole slide from a sample of the image blocks (tiles) that contain cells: `n_cells` cell positions are drawn, their storage blocks collected, `n_tiles` of these blocks drawn, every `step`-th pixel in both directions taken, and `sg_display_range()` applied per channel. Tiles follow the block grid of the file, so the estimate reads few blocks even for very large scans.

Usage

```
sg_slide_display_ranges(
  path,
  channels,
  cells,
  x = "x",
  y = "y",
  pixel_size = 1,
  n_cells = 4000,
  n_tiles = 60,
  step = 3,
  probs = c(0.01, 0.997),
  min_span = 1,
  seed = 0,
  source = NULL
)
```

Arguments

path	Path to the image file.
channels	Channel names or numbers.
cells	Data frame of cell positions in physical units.
x, y	Column names of the positions.
pixel_size	Physical size of one pixel.
n_cells, n_tiles	Numbers of cells and tiles drawn.
step	Pixel subsampling step inside each tile.
probs, min_span	Passed to sg_display_range().
seed	Random seed for the draws (the global RNG state is restored).
source	Optional label stored in a source column (for example the image name), so that ranges of several images can be stacked.

Value

A data frame with columns channel, lo, hi, tiles, pixels and, with source, source.

See Also

sg_display_levels()

Examples

```
f <- tempfile(fileext = ".tif")
planes <- list(outer(1:64, 1:64, function(i, j) (i + j) %% 200),
               outer(1:64, 1:64, function(i, j) (i * j) %% 250))
invisible(tiff::writeTIFF(lapply(planes, function(p) p / 255), f,
                          bits.per.sample = 8L))
cells <- data.frame(x = c(10, 30, 50, 60), y = c(12, 40, 55, 5))
sg_slide_display_ranges(f, c("ch1", "ch2"), cells, n_tiles = 2)
```

---

sg_stain_deconvolve *Separate H&E stain channels by colour deconvolution*

Description

Applies colour deconvolution to an H&E-stained image to separate the haematoxylin and eosin channels.

Usage

```
sg_stain_deconvolve(image, method = c("macenko", "ruifrok"), stains = NULL)
```

Arguments

image	An sg_image object with at least 3 colour channels (RGB).
method	Character string specifying the deconvolution method. One of "macenko" (default), which estimates the two stain vectors from the image, or "ruifrok", which uses the fixed haematoxylin and eosin vectors of Ruifrok and Johnston (2001).
stains	Optional 3x3 numeric matrix whose columns are the stain vectors (optical density in R, G and B) for haematoxylin, eosin and the residual channel, in that order. If NULL, stain vectors are estimated from the image using the specified method.

Value

A named list of sg_image objects, one per stain channel. Typically hematoxylin and eosin, plus residual.

Examples

```
# Requires a colour image
pixels <- array(runif(300), dim = c(10, 10, 3))
img <- new_sg_image(pixels, channels = c("R", "G", "B"))
stains <- sg_stain_deconvolve(img)
names(stains)
```

---

sg_stardist_capabilities

StarDist adapter capabilities

Description

Reports which StarDist options can be mapped for Python (reticulate) and for the QuPath StarDist extension, for a detected or declared QuPath and extension version. Detection reads installation files only (QuPath is not started). Options are supported for extension versions at or above the documented minimum, unknown for older versions and unavailable when the extension is missing.

Usage

```
sg_stardist_capabilities(
  qupath_version = NULL,
  extension_version = NULL,
  check_python = FALSE
)
```

Arguments

qupath_version	Optional QuPath version to evaluate (default: detected installation).
extension_version	Optional StarDist extension version (default: detected jar in the QuPath user directory).
check_python	Logical; probe the Python stardist module. Python is initialised only when one has been selected (see the section "Selecting Python" in sg_setup_python()).

Value

An sg_stardist_capabilities list with qupath, extension, python, options (tibble) and parameters (tibble from sg_stardist_parameter_map() for both targets).

Examples

```
caps <- sg_stardist_capabilities(qupath_version = "0.7.0",
                                 extension_version = "0.6.0")
caps$options[, c("option", "status")]
```

---

sg_stardist_manifest *Describe a StarDist model for exchange*

Description

Hashes the model files and records representation, channels, pixel size, normalisation, license and the runtimes that can load the representation (.pb for QuPath/OpenCV, SavedModel for QuPath/TensorFlow, bioimage.io for the extension's bioimage.io support, a Python model directory for stardist).

Usage

```
sg_stardist_manifest(
  model,
  ...,
  name = NULL,
  channels = NULL,
  n_channels_in = NULL,
  pixel_size_um = NULL,
  normalization = NULL,
  license_id = NULL,
  source = NULL
)
```

Arguments

model	Path to a .pb file, a SavedModel or Python model directory, a bioimage.io folder, or an sg_trained_model with backend "stardist".
...	Reserved; must be empty.
name	Model name (default: file/directory name).
channels	Optional channel names the model expects.
n_channels_in	Optional number of input channels.
pixel_size_um	Optional pixel size the model was trained for.
normalization	Optional list (e.g. list(method = "percentile", low = 1, high = 99.8)).
license_id	Optional SPDX identifier.
source	Optional short provenance text (no URLs are fetched).

Value

An sg_stardist_manifest list conforming to stardist-model.schema.json; digest follows the integrity inventory rule and is what run.json and the Groovy template verify.

Examples

```
f <- tempfile(fileext = ".pb")
writeBin(as.raw(1:10), f)
sg_stardist_manifest(f, n_channels_in = 1L)$representation
```

---

sg_stardist_parameter_map

Map StarDist protocol parameters to Python or QuPath

Description

Every declared parameter of the protocol appears in the result with the target option, the target value and a status: mapped, approximated, unsupported, not_applicable or runner (handled by segmantR before delegation). Unknown parameters are errors; nothing is dropped silently.

Usage

```
sg_stardist_parameter_map(
  protocol = "stardist.2d.v1",
  target = c("python", "qupath"),
  ...,
  image = NULL
)
```

Arguments

protocol	Protocol id or sg_protocol (StarDist family).
target	"python" (reticulate, sg_segment_stardist()) or "qupath" (StarDist extension builder).
...	Parameter overrides.
image	Optional sg_image used to resolve channel indices and wavelengths to channel names for QuPath.

Value

A tibble with parameter, value, target, target_option, target_value, status and note; attribute parameters holds the resolved parameters.

Examples

```
sg_stardist_parameter_map("stardist.2d.v1", "qupath", prob_thresh = 0.6)
```

---

sg_train_cellpose *Fine-tune a Cellpose model on user-corrected masks*

Description

Trains (fine-tunes) a Cellpose segmentation model using corrected image-mask pairs provided by the user. Requires a working Python environment with Cellpose installed (see sg_setup_python()).

Usage

```
sg_train_cellpose(
  training_data,
  base_model = "cyto3",
  n_epochs = 100L,
  learning_rate = 0.1,
  save_path = NULL,
  verbose = TRUE
)
```

Arguments

training_data	A list of training pairs. Each element must be a list with $image (sg_image) and $mask (sg_mask).
base_model	Character. Cellpose base model to fine-tune. Default "cyto3".
n_epochs	Integer. Number of training epochs. Default 100L.
learning_rate	Numeric. Learning rate. Default 0.1.
save_path	Character or NULL. Directory to save the trained model. When NULL, a temporary directory is used.
verbose	Logical. Print training progress? Default TRUE.

Value

An sg_trained_model object.

Examples

```
## Not run:
# Requires Python with Cellpose (see sg_setup_python())
pair <- list(image = sg_example_image("fluorescence_nuclei"),
             mask = sg_example_mask("fluorescence_nuclei"))
trained <- sg_train_cellpose(list(pair), base_model = "cyto3",
                             n_epochs = 10L)

## End(Not run)
```

---

sg_train_stardist *Train a StarDist 2D model*

Description

Trains a StarDist model on a prepared instance dataset. All data checks run in R before Python is touched: the dataset must be an instance dataset without split leakage, every training mask must be integer with background 0 and one connected part per positive id, and the training split must not be empty. Only then is the optional Python runtime (stardist, tensorflow, csbdeep through reticulate) checked; when it is missing an sg_capability_error with code CAPABILITY_UNAVAILABLE and an installation hint is signalled.

Usage

```
sg_train_stardist(
  training_data,
  base_model = NULL,
  n_epochs = 100L,
  steps_per_epoch = 100L,
  learning_rate = 3e-04,
  n_rays = 32L,
  grid = c(2L, 2L),
  normalize = c(1, 99.8),
  save_path = NULL,
  name = "stardist_model",
  model_card = list(),
  seed = 1L,
  verbose = TRUE
)
```

Arguments

training_data	An sg_training_set from sg_prepare_training_data() (in memory or on disk), or the path of a prepared dataset directory.
base_model	Optional pretrained StarDist model name to fine-tune (e.g. "2D_versatile_fluo"); NULL trains from a new configuration.
n_epochs, steps_per_epoch	Training length.
learning_rate	Adam learning rate.
n_rays	Number of radial directions.
grid	Subsampling grid (length 2).
normalize	Percentiles (length 2) for per-tile normalisation, recorded in the model card.
save_path	Directory for the trained model (default: a temporary directory).
name	Model name (directory name below save_path).
model_card	Model card fields (purpose, data_domain, limitations, license, ...), see new_sg_trained_model().
seed	Integer seed passed to the Python random number generators.
verbose	Logical; print progress.

Value

An sg_trained_model with backend "stardist", the dataset manifest, runtime description and evaluation metrics (validation split thresholds when available).

Examples

```
img <- sg_example_image("fluorescence_nuclei")
msk <- sg_cleanup_labels(sg_example_mask("fluorescence_nuclei"),
                         disconnected = "split")
ts <- sg_prepare_training_data(
  list(list(image = img, mask = msk, subject_id = "A"),
       list(image = img, mask = msk, subject_id = "B")),
  tile_size = 32L, split = c(train = 0.5, validation = 0.5, test = 0)
)
# Requires Python stardist; otherwise a classified capability error:
try(sg_train_stardist(ts, n_epochs = 1L))
```

---

sg_validate_interchange

Validate interchange data

Description

Validates a bundle or file on disk (all checks of sg_import_interchange()) or an in-memory object: sg_manifest or list (JSON Schema and version), sg_mask (label contract, legend completeness, review consistency), sg_run (run schema) or sg_protocol.

Usage

```
sg_validate_interchange(path_or_object, ...)
```

Arguments

path_or_object	A path, sg_manifest, manifest list, sg_mask, sg_run or sg_protocol.
...	Passed to sg_import_interchange() for paths.

Value

An sg_validation_report with ok, checks (tibble) and error.

Examples

```
m <- sg_example_mask("fluorescence_nuclei")
sg_validate_interchange(m)$ok
bad <- m
bad$labels[1, 1] <- -3L
sg_validate_interchange(bad)$ok
```

---

sg_validate_training_manifest

Validate a training dataset manifest

Description

Checks schema conformance, split leakage (a group or source in more than one split), tile bounds, exclusion bookkeeping, instance/label contract of materialised mask tiles (integer dtype, background 0, unique connected instance ids, complete object lists), recorded tile digests, the dataset digest and, for directories, the integrity inventory.

Usage

```
sg_validate_training_manifest(
  x,
  mask_type = NULL,
  check_files = TRUE,
  error = TRUE
)
```

Arguments

x	An sg_dataset_manifest, sg_training_set, a prepared dataset directory or a dataset_manifest.json path.
mask_type	Optional mask type the consumer requires (e.g. "instance" for StarDist).
check_files	Logical; read and check tile files of a directory.
error	Logical; signal the first failure instead of reporting it.

Value

An sg_validation_report with ok, checks and error.

Examples

```
img <- sg_example_image("fluorescence_nuclei")
# The synthetic example mask contains overlapping ellipses; split them
# into separate instances first.
msk <- sg_cleanup_labels(sg_example_mask("fluorescence_nuclei"),
                         disconnected = "split")
man <- sg_dataset_manifest(list(list(image = img, mask = msk,
                                     subject_id = "A")),
                           tile_size = 32L, min_objects = 0L,
                           split = c(train = 1, validation = 0, test = 0))
sg_validate_training_manifest(man)$ok
```

---

sgCanvasOutput *Shiny bindings for the cell canvas*

Description

Shiny bindings for the cell canvas

Usage

```
sgCanvasOutput(outputId, width = "100%", height = "600px")

renderSgCanvas(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

outputId	Output id.
width, height	Output dimensions.
expr	Expression returning sg_canvas().
env	Evaluation environment.
quoted	Whether expr is quoted.

Value

A Shiny output element or render function.

Examples

```
if (interactive() && requireNamespace("htmlwidgets", quietly = TRUE) &&
    requireNamespace("magick", quietly = TRUE)) {
  img <- new_sg_image(matrix(seq_len(72), 8, 9))
  ui <- shiny::fluidPage(sgCanvasOutput("canvas"))
  server <- function(input, output, session) {
    output$canvas <- renderSgCanvas(sg_canvas(img))
  }
  shiny::shinyApp(ui, server)
}
```

Index

∗ **geometry**
  sg_roi_buffer, 62
  sg_roi_check_containment, 63
  sg_roi_check_geometry, 64
  sg_roi_clamp, 66
  sg_roi_contains, 67
  sg_roi_distance, 68
  sg_roi_fix_geometry, 70
  sg_roi_flip_y, 71
  sg_roi_overlaps, 86
  sg_roi_rescale, 94
  sg_roi_ring, 95
  sg_roi_rois_overlap, 97
  sg_roi_setops, 98
  sg_roi_simplify, 99
  sg_roi_snap, 100
  sg_roi_transform, 102
∗ **images**
  sg_roi_dims, 67
∗ **io**
  sg_roi_from_geojson, 73
  sg_roi_read_geojson, 89
  sg_roi_read_qupath, 90
  sg_roi_write_geojson, 104
  sg_roi_write_qupath, 105
∗ **layers**
  sg_roi_layer, 75
  sg_roi_layer_add, 76
  sg_roi_layer_n, 77
  sg_roi_layer_remove, 77
  sg_roi_layer_rois, 78
  sg_roi_style, 100
∗ **masks**
  sg_roi_mask, 79
  sg_roi_mask_boundary, 81
  sg_roi_mask_derive, 82
  sg_roi_mask_legend, 83
  sg_roi_mask_preview, 84
  sg_roi_mask_stack, 84
  sg_roi_mask_stats, 85
∗ **projects**
  sg_roi_add_layer, 59
  sg_roi_add_roi, 59
  sg_roi_layers, 79
  sg_roi_project, 89
  sg_roi_remove_layer, 93
  sg_roi_remove_roi, 93
  sg_roi_rois, 96
  sg_roi_summary, 102
  sg_roi_validate, 103
∗ **rois**
  sg_roi_area, 60
  sg_roi_bbox, 61
  sg_roi_centroid, 62
  sg_roi_circle, 65
  sg_roi_ellipse, 69
  sg_roi_freehand, 72
  sg_roi_from_geojson, 73
  sg_roi_from_sf, 74
  sg_roi_point, 87
  sg_roi_polygon, 88
  sg_roi_rect, 92

cli::cli_inform(), *70*

new_sg_image, 5
new_sg_image(), *7*, *114*
new_sg_mask, 6
new_sg_trained_model, 8
new_sg_trained_model(), *123*

renderSgCanvas *(*sgCanvasOutput*)*, 125

sg_active_learning_loop, 9
sg_app, 10
sg_app(), *106*
sg_apply_corrections, 11
sg_canvas, 12
sg_canvas(), *125*

sg_canvas_correct, 13
sg_canvas_correct(), *12*
sg_cleanup_labels, 14
sg_control_capabilities, 15
sg_control_capabilities(), *10*
sg_create_annotation_task, 16
sg_dataset_manifest, 17
sg_dataset_manifest(), *48*
sg_display_levels, 18
sg_display_levels(), *20*, *117*
sg_display_range, 20
sg_display_range(), *19*, *116, 117*
sg_envi_info, 21
sg_evaluate_segmentation, 21
sg_example_image, 22
sg_example_image(), *23*
sg_example_mask, 23
sg_example_mask(), *23*
sg_export_interchange, 24
sg_export_interchange(), *27*, *35*, *51*
sg_export_mask, 25
sg_export_qupath, 26
sg_extract_features, 27
sg_filter_cells, 28
sg_hash_assets, 29
sg_image_channels, 30
sg_image_channels(), *58*
sg_import_interchange, 31
sg_import_interchange(), *32*, *123, 124*
sg_import_qupath, 32
sg_interchange_capabilities, 33
sg_interchange_manifest, 34
sg_load_model, 35
sg_mask_legend, 36
sg_mask_legend(), *7*
sg_mask_review, 37
sg_mask_revision, 38
sg_mask_status, 39
sg_mask_to_polygons, 39
sg_merge_masks, 40
sg_model_card, 41
sg_package_model, 41
sg_package_model(), *8*, *35*
sg_plot_comparison, 43
sg_plot_features, 44
sg_plot_mask, 45
sg_plot_metrics, 45
sg_plot_overlay, 46
sg_predict_model, 47
sg_prepare_training_data, 48
sg_prepare_training_data(), *123*
sg_preprocess, 49
sg_protocol_get, 50
sg_protocol_list, 50
sg_protocol_list(), *51*
sg_protocol_run, 51
sg_protocol_run(), *47*
sg_protocol_schema, 52
sg_protocol_schema(), *51*
sg_protocol_validate, 52
sg_protocol_validate(), *51*
sg_read_envi, 53
sg_read_envi(), *114*
sg_read_field, 54
sg_read_field(), *31*, *55, 56*, *58*
sg_read_fields, 55
sg_read_fields(), *55*
sg_read_image, 57
sg_read_window, 57
sg_read_window(), *31*, *55*
sg_replace_mask *(*sg_mask_review*)*, 37
sg_replace_mask(), *31*, *47*
sg_review_mask *(*sg_mask_review*)*, 37
sg_review_mask(), *7*
sg_roi, 58, *60–63*, *65–69*, *71*, *73–76*, *80*, *86–88*, *92*, *94, 95*, *98–100*, *103*
sg_roi_add_layer, 59
sg_roi_add_layer(), *60*, *79*, *89*, *93, 94*, *97*, *102*, *104*
sg_roi_add_roi, 59
sg_roi_add_roi(), *59*, *79*, *89*, *93, 94*, *97*, *102*, *104*
sg_roi_area, 60
sg_roi_area(), *61*, *63*, *65*, *70*, *73–75*, *87, 88*, *92*
sg_roi_bbox, 61
sg_roi_bbox(), *61*, *63*, *65*, *70*, *73–75*, *87, 88*, *92*
sg_roi_buffer, 62
sg_roi_buffer(), *64*, *66–68*, *70, 71*, *87*, *95, 96*, *98–100*, *103*
sg_roi_centroid, 62
sg_roi_centroid(), *61*, *65*, *70*, *73–75*, *87, 88*, *92*
sg_roi_check_containment, 63
sg_roi_check_containment(), *62*, *64*,

66–68, *70, 71*, *87*, *95, 96*, *98–100*, *103*
sg_roi_check_geometry, 64
sg_roi_check_geometry(), *62–64*, *66–68*, *70, 71*, *87*, *95, 96*, *98–100*, *103*
sg_roi_circle, 65
sg_roi_circle(), *61*, *63*, *70*, *73–75*, *87, 88*, *92*
sg_roi_clamp, 66
sg_roi_clamp(), *62*, *64*, *67, 68*, *70, 71*, *87*, *95, 96*, *98–100*, *103*
sg_roi_contains, 67
sg_roi_contains(), *62*, *64*, *66*, *68*, *70, 71*, *87*, *95, 96*, *98–100*, *103*
sg_roi_difference *(*sg_roi_setops*)*, 98
sg_roi_difference(), *96*
sg_roi_dims, 67
sg_roi_distance, 68
sg_roi_distance(), *62*, *64*, *66, 67*, *70, 71*, *87*, *95, 96*, *98–100*, *103*
sg_roi_ellipse, 69
sg_roi_ellipse(), *61*, *63*, *65*, *73–75*, *87, 88*, *92*
sg_roi_fix_geometry, 70
sg_roi_fix_geometry(), *62*, *64*, *66–68*, *71*, *87*, *95, 96*, *98–100*, *103*
sg_roi_flip_y, 71
sg_roi_flip_y(), *62*, *64*, *66–68*, *70*, *87*, *95, 96*, *98–100*, *103*
sg_roi_frame, *67, 68*, 71, *78*, *89*, *94*, *102, 103*
sg_roi_frame(), *58*, *68*
sg_roi_freehand, 72
sg_roi_freehand(), *61*, *63*, *65*, *70*, *74, 75*, *87, 88*, *92*
sg_roi_from_geojson, 73
sg_roi_from_geojson(), *61*, *63*, *65*, *70*, *73*, *75*, *87, 88*, *90–92*, *105, 106*
sg_roi_from_sf, 74
sg_roi_from_sf(), *61*, *63*, *65*, *70*, *73, 74*, *87, 88*, *92*
sg_roi_intersect *(*sg_roi_setops*)*, 98
sg_roi_layer, *59*, 75, *75–78*, *80*, *89–91*, *97, 98*, *100*
sg_roi_layer(), *58*, *76–78*, *101*
sg_roi_layer_add, 76
sg_roi_layer_add(), *75*, *77, 78*, *101*
sg_roi_layer_n, 77
sg_roi_layer_n(), *75, 76*, *78*, *101*
sg_roi_layer_remove, 77
sg_roi_layer_remove(), *75–78*, *101*
sg_roi_layer_rois, 78
sg_roi_layer_rois(), *75–78*, *101*
sg_roi_layers, 79
sg_roi_layers(), *59, 60*, *89*, *93, 94*, *97*, *102*, *104*
sg_roi_mask, 79
sg_roi_mask(), *58*, *81–86*
sg_roi_mask_boundary, 81
sg_roi_mask_boundary(), *81–86*
sg_roi_mask_derive, 82
sg_roi_mask_derive(), *81*, *83–86*
sg_roi_mask_legend, 83
sg_roi_mask_legend(), *81, 82*, *84–86*
sg_roi_mask_preview, 84
sg_roi_mask_preview(), *81–83*, *85, 86*
sg_roi_mask_stack, 84
sg_roi_mask_stack(), *81–84*, *86*
sg_roi_mask_stats, 85
sg_roi_mask_stats(), *81–85*
sg_roi_overlaps, 86
sg_roi_overlaps(), *62*, *64*, *66–68*, *70, 71*, *95, 96*, *98–100*, *103*
sg_roi_point, 87
sg_roi_point(), *61*, *63*, *65*, *70*, *73–75*, *88*, *92*
sg_roi_polygon, 88
sg_roi_polygon(), *61*, *63*, *65*, *70*, *73–75*, *87*, *92*
sg_roi_project, *59, 60*, *63, 64*, *70*, *79, 80*, *85*, 89, *89*, *93, 94*, *96*, *102–105*
sg_roi_project(), *58–60*, *72*, *79*, *93, 94*, *97*, *102*, *104*
sg_roi_read_geojson, 89
sg_roi_read_geojson(), *74*, *91*, *104–106*
sg_roi_read_qupath, 90
sg_roi_read_qupath(), *74*, *90*, *105, 106*
sg_roi_rect, 92
sg_roi_rect(), *58*, *61*, *63*, *65*, *70*, *73–75*, *87, 88*
sg_roi_remove_layer, 93
sg_roi_remove_layer(), *59, 60*, *79*, *89*, *94*, *97*, *102*, *104*
sg_roi_remove_roi, 93
sg_roi_remove_roi(), *59, 60*, *79*, *89*, *93*, *97*, *102*, *104*
sg_roi_rescale, 94
sg_roi_rescale(), *62*, *64*, *66–68*, *70, 71*, *87*,

96, *98–100*, *102*, *103*
sg_roi_ring, 95
sg_roi_ring(), *62*, *64*, *66–68*, *70*, *71*, *87*, *95*, *98–100*, *103*
sg_roi_rois, 96
sg_roi_rois(), *59*, *60*, *78*, *79*, *89*, *93*, *94*, *102*, *104*
sg_roi_rois_overlap, 97
sg_roi_rois_overlap(), *62*, *64*, *66–68*, *70*, *71*, *87*, *95*, *96*, *98–100*, *103*
sg_roi_setops, *62*, *64*, *66–68*, *70*, *71*, *87*, *95*, *96*, 98, *98–100*, *103*
sg_roi_simplify, 99
sg_roi_simplify(), *62*, *64*, *66–68*, *70*, *71*, *87*, *95*, *96*, *98*, *100*, *103*
sg_roi_snap, 100
sg_roi_snap(), *62*, *64*, *66–68*, *70*, *71*, *87*, *95*, *96*, *98*, *99*, *103*
sg_roi_style, 100
sg_roi_style(), *75–78*
sg_roi_summary, 102
sg_roi_summary(), *59*, *60*, *79*, *89*, *93*, *94*, *97*, *104*
sg_roi_symdiff *(*sg_roi_setops*)*, 98
sg_roi_transform, 102
sg_roi_transform(), *62*, *64*, *66–68*, *70–72*, *87*, *95*, *96*, *98–100*
sg_roi_union *(*sg_roi_setops*)*, 98
sg_roi_validate, 103
sg_roi_validate(), *59*, *60*, *64*, *79*, *89*, *93*, *94*, *97*, *102*
sg_roi_write_geojson, 104
sg_roi_write_geojson(), *74*, *90*, *91*, *106*
sg_roi_write_qupath, 105
sg_roi_write_qupath(), *74*, *90*, *91*, *105*
sg_run_app, 106
sg_run_app(), *10*
sg_segment_cellpose, 107
sg_segment_mesmer, 108
sg_segment_propagate, 109
sg_segment_stardist, 110
sg_segment_stardist(), *121*
sg_segment_threshold, 111
sg_segment_watershed, 112
sg_select_channel, 114
sg_setup_python, 115
sg_setup_python(), *34*, *50*, *107*, *110*, *119*, *121*
sg_slide_display_ranges, 116
sg_slide_display_ranges(), *19*, *20*
sg_stage_mask *(*sg_mask_review*)*, 37
sg_stain_deconvolve, 117
sg_stardist_capabilities, 118
sg_stardist_manifest, 119
sg_stardist_parameter_map, 120
sg_stardist_parameter_map(), *27*, *119*
sg_train_cellpose, 121
sg_train_stardist, 122
sg_validate_interchange, 123
sg_validate_training_manifest, 124
sgCanvasOutput, 125

tibble::tibble, *63*, *64*, *79*, *83*, *86*, *96*, *97*