

Package: cellspecR (via r-universe)

October 29, 2026

Title Read, Validate and Store Multiplexed Imaging Cell Tables

Version 1.1.0

Description Defines 'cellspec', a versioned table format for segmented cells from multiplexed tissue images: cell identifiers, centroids in micrometres, per-marker measurements with an explicit compartment and statistic, image and channel metadata, and provenance. Reads measurement exports of image-analysis tools such as 'QuPath' and 'MCMICRO' into this format, validates its structure and content, selects one signal per marker through an explicit policy, and writes directories with a 'SHA-256' checksum manifest so that later analysis steps can verify what they read. 'QuPath' is described by Bankhead et al. (2017) <[doi:10.1038/s41598-017-17204-5](https://doi.org/10.1038/s41598-017-17204-5)> and 'MCMICRO' by Schapiro et al. (2022) <[doi:10.1038/s41592-021-01308-y](https://doi.org/10.1038/s41592-021-01308-y)>.

License MIT + file LICENSE

URL <https://github.com/CTTIR/cellspecR>,
<https://cttir.github.io/cellspecR/>

BugReports <https://github.com/CTTIR/cellspecR/issues>

Depends R (>= 4.1.0)

Imports cli, data.table (>= 1.17.0), digest, jsonlite, nanoparquet,
rlang, stats, utils

Suggests anndataR, bslib, ggplot2, jsonvalidate, knitr, patchwork,
rmarkdown, S4Vectors, shiny, SingleCellExperiment,
SpatialExperiment, SummarizedExperiment, testthat (>= 3.1.7),
tibble, withr

VignetteBuilder knitr

Config/testthat/edition 3

Config/Needs/dev bench

Config/Needs/website CTTIR/themakR

Encoding UTF-8

Language en-GB

Roxygen list(markdown = TRUE)
Config/roxygen2/version 8.0.0
Repository https://cttir.r-universe.dev
Date/Publication 2026-10-29 11:19:01 UTC
RemoteUrl https://github.com/CTTIR/cellspecR
RemoteRef HEAD
RemoteSha bd940f1baf937cd6a78f8741193ac419d34c712b

Contents

[.cellspec	3
as.data.frame.cellspec	4
cellspec-methods	5
cs_accessors	6
cs_app	7
cs_as_anndata	8
cs_as_spe	9
cs_bind	10
cs_column_map	10
cs_detect_format	11
cs_example	12
cs_feature_support	13
cs_features	14
cs_formats	15
cs_from_spe	15
cs_marker_map	16
cs_merge_tiles	17
cs_new	18
cs_read	19
cs_read_cellspec	21
cs_read_npy	22
cs_read_signal_policy	23
cs_signal_matrix	24
cs_signal_policy	25
cs_spec_version	26
cs_validate	27
cs_verify	28
cs_vocabulary	29
cs_write	30
cs_write_signal_policy	31
is_cellspec	31
plot.cellspec	32

Index	34
--------------	-----------

Description

`x[i, j]` keeps the cells selected by `i` and the features selected by `j`. All components stay aligned: cells and measurements rows are subset together, dictionary rows follow the kept features, and adjacency contacts whose cells were removed are dropped. images and channels are kept unchanged.

Usage

```
## S3 method for class 'cellspec'
x[i, j, ..., drop = FALSE]
```

Arguments

<code>x</code>	A cellspec object.
<code>i</code>	Cells to keep: a logical vector with one value per cell, or positive or negative integer positions. Missing means all cells.
<code>j</code>	Features to keep: a character vector of feature identifiers, a logical vector with one value per feature, or integer positions. Missing means all features.
<code>...</code>	Not used; must be empty.
<code>drop</code>	Not used; subsetting always returns a cellspec object.

Value

A cellspec object.

See Also

Other methods: [as.data.frame.celspec\(\)](#), [cellspec-methods](#), [plot.celspec\(\)](#)

Examples

```
x <- cs_example()
dim(x)
big <- x[cs_cells(x)$area > 60, ]
dim(big)
means <- x[, cs_features(x, statistic = "mean")]
dim(means)
```

`as.data.frame.cellspec`*Convert a cellspec object to a data frame*

Description

Combines the cells columns with selected measurement columns into one wide data frame, for example to hand a table to modelling code. Measurement columns are named by their feature identifiers.

Usage

```
## S3 method for class 'cellspec'
as.data.frame(x, row.names = NULL, optional = FALSE, ..., features = NULL)

## S3 method for class 'cellspec'
as_tibble(x, ..., features = NULL)
```

Arguments

<code>x</code>	A cellspec object.
<code>row.names, optional</code>	Ignored; present for compatibility with the generic.
<code>...</code>	Unused.
<code>features</code>	Character vector of feature identifiers to include (default: all features).

Value

A data.frame with one row per cell: the cells columns followed by one double column per selected feature. `as_tibble()` returns the same content as a tibble (requires the tibble package).

See Also

Other methods: [\[.cellspec\(\)](#), [cellspec-methods](#), [plot.celspec\(\)](#)

Examples

```
x <- cs_example()
df <- as.data.frame(x, features = cs_features(x, statistic = "mean"))
head(df)
```

cellspec-methods

*Print, summarise and measure a cellspec object***Description**

`print()` shows a compact overview that fits in 80 columns: cells, images, samples, features by kind, markers, pixel size, specification version and the adapter that produced the object. `summary()` returns one row per image. `dim()` returns the number of cells and features.

Usage

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

## S3 method for class 'cellspec'
summary(object, ...)

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

## S3 method for class 'cellspec'
dim(x)
```

Arguments

<code>x, object</code>	A cellspec object (or, for the summary print method, a cellspec_summary).
<code>...</code>	Unused; for compatibility with the generics.

Value

- `print()` returns `x` invisibly.
- `summary()` returns a data frame of class `cellspec_summary` with one row per image and the columns `image_id`, `sample_id` (character), `n_cells`, `n_markers`, `n_features` (integer; markers and features with at least one non-missing value in that image), `pixel_size`, `x_min`, `x_max`, `y_min`, `y_max` (double, micrometres) and `na_fraction` (double, fraction of missing measurement values in that image).
- `dim()` returns an integer vector `c(cells, features)`.

See Also

Other methods: [\[.cellspec\(\)](#), [as.data.frame.celspec\(\)](#), [plot.celspec\(\)](#)

Examples

```
x <- cs_example()
x
summary(x)
dim(x)
```

cs_accessors

Access the components of a cellspec object

Description

Accessors return the components of a cellspec object. Use them instead of `x$cells` so code keeps working if the internal layout changes.

Usage

`cs_cells(x)`

`cs_measurements(x)`

`cs_dictionary(x)`

`cs_images(x)`

`cs_channels(x)`

`cs_provenance(x)`

`cs_adjacency(x)`

Arguments

`x` A cellspec object.

Value

- `cs_cells()`: data frame, one row per cell.
- `cs_measurements()`: double matrix, cells x features, column names are feature identifiers.
- `cs_dictionary()`: data frame, one row per measurement column.
- `cs_images()`: data frame, one row per image.
- `cs_channels()`: data frame, one row per image and channel.
- `cs_provenance()`: list with producer, reader, created_utc, inputs, counts, parameters and history.
- `cs_adjacency()`: data frame of cell-cell contacts, or NULL.

See Also

`vignette("specification", package = "cellspecR")`

Other object: [cs_bind\(\)](#), [cs_example\(\)](#), [cs_features\(\)](#), [cs_merge_tiles\(\)](#), [cs_new\(\)](#), [is_cellspec\(\)](#)

Examples

```
x <- cs_example()
head(cs_cells(x))
dim(cs_measurements(x))
cs_dictionary(x)[1:3, ]
cs_images(x)
cs_channels(x)
names(cs_provenance(x))
cs_adjacency(x)
```

cs_app

*Create the cellspec review application***Description****[Stable]**

Builds a local Shiny application for reading an export, reviewing format detection and validation, and downloading the resulting tables. The function returns an application object; launch it with `shiny::runApp()`.

Usage

```
cs_app(
  path = NULL,
  max_upload_mb = 2048,
  allow_local_paths = FALSE,
  max_plot_points = 50000L,
  ...
)
```

Arguments

path	Optional initial file or directory path. The user can also select a file in the application.
max_upload_mb	Maximum upload size in MB. It is applied as the <code>shiny.maxRequestSize</code> option while the application runs and the previous value is restored when it stops.
allow_local_paths	Whether a local path field is available.
max_plot_points	Maximum cells passed to plots.
...	Reserved for launcher settings.

Value

A `shiny.appobj`.

Examples

```
if (requireNamespace("shiny", quietly = TRUE)) {
  app <- cs_app()
  inherits(app, "shiny.appobj")
}
```

cs_as_anndata

Convert a cellspec object to AnnData

Description

Creates an AnnData object with cells in obs, selected marker signals in X, and x/y in obsm\$spatial. The optional anndataR package and its Python runtime are required.

Usage

```
cs_as_anndata(x, policy = NULL)
```

Arguments

x	A cellspec object.
policy	Signal policy used to form X. It is required when a marker has more than one compartment or statistic; otherwise it can be omitted.

Value

An AnnData object supplied by anndataR.

See Also

Other interoperability: [cs_as_spe\(\)](#), [cs_from_spe\(\)](#)

Examples

```
if (requireNamespace("anndataR", quietly = TRUE)) {
  x <- cs_example()
  cs_as_anndata(x, cs_signal_policy(cs_markers(x)))
}
```

cs_as_spe	<i>Convert a cellspec object to SpatialExperiment</i>
-----------	---

Description

Converts selected signals to the main assay, stores every raw measurement in an `altExp`, and records the cellspec metadata needed for a lossless round trip.

Usage

```
cs_as_spe(x, policy = NULL, assay_name = "intensity")
```

Arguments

x	A cellspec object.
policy	A signal policy. It is required when a marker has more than one compartment or statistic; otherwise it can be omitted.
assay_name	Name of the main signal assay.

Value

A `SpatialExperiment` with cells as columns, markers as rows, `spatialCoords` containing x and y, and a `measurements altExp`.

See Also

Other interoperability: [cs_as_anndata\(\)](#), [cs_from_spe\(\)](#)

Examples

```
if (requireNamespace("SpatialExperiment", quietly = TRUE)) {  
  x <- cs_example()  
  policy <- cs_signal_policy(cs_markers(x))  
  spe <- cs_as_spe(x, policy)  
  spe  
}
```

cs_bind	<i>Combine cellspec objects from different images</i>
---------	---

Description

cs_bind() combines cell tables, measurements and metadata from one or more cellspec objects. Objects must describe different images and must use the same measurement dictionary.

Usage

```
cs_bind(...)
```

Arguments

... cellspec objects to combine.

Value

A combined cellspec object.

See Also

Other object: [cs_accessors](#), [cs_example\(\)](#), [cs_features\(\)](#), [cs_merge_tiles\(\)](#), [cs_new\(\)](#), [is_cellspec\(\)](#)

Examples

```
cs_bind(cs_example())
```

cs_column_map	<i>Validate a table column map</i>
---------------	------------------------------------

Description

Creates a validated description of how a generic delimited table maps to the cellspec columns. Source names are checked for shape here and checked against the input table by [cs_read\(\)](#).

Usage

```
cs_column_map(
  cell_id,
  x,
  y,
  image_id = NULL,
  sample_id = NULL,
  area = NULL,
```

```

    coordinate_unit = c("um", "px"),
    measurements = NULL
  )

```

Arguments

cell_id	Source column containing the cell identifier.
x	Source column containing the x coordinate.
y	Source column containing the y coordinate.
image_id	Optional source column containing image identifiers.
sample_id	Optional source column containing sample identifiers.
area	Optional source column containing positive cell areas.
coordinate_unit	Coordinate unit for x and y: micrometres or pixels.
measurements	Optional data frame with source_name, kind, marker, compartment, statistic and unit metadata columns. Measurements of kind other get other: identifiers as described for keep_other in cs_read() .

Value

An object of class `cs_column_map`, a named list with the mapped source names, coordinate unit and validated measurement metadata.

See Also

[cs_read\(\)](#)

Other adapters: [cs_detect_format\(\)](#), [cs_formats\(\)](#), [cs_read\(\)](#)

Examples

```

cs_column_map(
  "Cell ID", "Centroid X", "Centroid Y",
  measurements = data.frame(
    source_name = "CD3e Mean", kind = "intensity", marker = "CD3e",
    compartment = "cell", statistic = "mean", unit = "a.u."
  )
)

```

cs_detect_format

Detect the generic table format

Description

Reads only the header and at most `n_max` data rows. A generic table is deliberately returned as a low-confidence candidate because its biological meaning cannot be inferred safely without an explicit column map.

Usage

```
cs_detect_format(path, n_max = 50L)
```

Arguments

`path` A delimited table file.

`n_max` Maximum number of data rows to inspect after the header.

Value

A data frame with format, score and evidence columns.

See Also

[cs_column_map\(\)](#), [cs_read\(\)](#)

Other adapters: [cs_column_map\(\)](#), [cs_formats\(\)](#), [cs_read\(\)](#)

Examples

```
path <- tempfile(fileext = ".csv")
utils::write.csv(data.frame(id = 1, x = 2, y = 3), path, row.names = FALSE)
cs_detect_format(path)
```

cs_example

A small synthetic cellspec object

Description

Builds a small, deterministic cellspec object for examples and experiments: two images from two samples, 40 cells each, four markers (DAPI, CD3e, Pan-Cytokeratin, FOXP3) measured as mean and median in the cell and nucleus compartments, cell and nucleus areas, and one tool-specific column. The values are simulated, not measured. The random number generator state of the session is left untouched.

Usage

```
cs_example()
```

Value

A valid cellspec object with 80 cells and 19 features.

See Also

Other object: [cs_accessors](#), [cs_bind\(\)](#), [cs_features\(\)](#), [cs_merge_tiles\(\)](#), [cs_new\(\)](#), [is_cellspec\(\)](#)

Examples

```
x <- cs_example()
x
cs_validate(x)
```

cs_feature_support	<i>Summarise the signal support of features per image</i>
--------------------	---

Description

Describes, for every image and feature, how much usable variation a feature carries. A marker whose channel was blank, saturated or not acquired shows up as "constant" or "near_constant" here before any normalisation or gating hides the problem. Other packages (for example a gating package) use the support column to declare markers non-callable.

The support classes are assigned in this order:

1. "mostly_na": more than half of the values are missing.
2. "constant": all non-missing values are identical (for example an all-zero channel).
3. "near_constant": the 1st and 99th percentiles are identical, so at most about 1% of cells differ from the rest.
4. "ok": otherwise.

Usage

```
cs_feature_support(
  x,
  features = cs_features(x, kind = "intensity", statistic = "mean"),
  zero_tol = 0
)
```

Arguments

x	A cellspec object.
features	Character vector of feature identifiers. Defaults to all mean intensity features.
zero_tol	Values with absolute value at most zero_tol count as zero for fraction_zero.

Value

A data frame with one row per image and feature and the columns image_id, feature_id, marker, compartment, statistic (character), n, n_na (integer), fraction_zero, sd, mad, q01, q50, q99 (double, computed on non-missing values; NA when none) and support (character, see Details).

See Also

Other validation: [cs_validate\(\)](#)

Examples

```
x <- cs_example()
cs_feature_support(x)

# A blank channel is reported as constant
x$measurements[, "cell:FOXP3:mean"] <- 0
s <- cs_feature_support(x)
s[s$marker == "FOXP3", c("image_id", "feature_id", "support")]
```

cs_features

Select features and markers by their dictionary entries

Description

cs_features() returns the identifiers of measurement columns whose dictionary entries match every supplied filter. Select features through this function rather than by splitting feature_id strings; the identifier layout is not part of the contract downstream code may parse.

cs_markers() returns the markers that have at least one intensity feature.

Usage

```
cs_features(
  x,
  kind = NULL,
  marker = NULL,
  compartment = NULL,
  statistic = NULL
)

cs_markers(x)
```

Arguments

x A cellspec object.

kind, marker, compartment, statistic Character vectors of allowed values, or NULL for no filter on that column.

Value

cs_features(): character vector of feature_id values in dictionary order (length 0 when nothing matches). cs_markers(): character vector of marker names in order of first appearance.

See Also

Other object: [cs_accessors](#), [cs_bind\(\)](#), [cs_example\(\)](#), [cs_merge_tiles\(\)](#), [cs_new\(\)](#), [is_cellspec\(\)](#)

Examples

```
x <- cs_example()
cs_features(x, kind = "intensity", statistic = "mean")
cs_features(x, marker = "FOXP3")
cs_markers(x)
```

cs_formats

List the available cellspec readers

Description

Lists the adapters implemented by this build of cellspecR. The generic table adapter accepts delimited files when the source columns are mapped explicitly with [cs_column_map\(\)](#).

Usage

```
cs_formats()
```

Value

A data frame with format, adapter_version, description, tested_with and status columns.

See Also

[cs_detect_format\(\)](#), [cs_column_map\(\)](#), [cs_read\(\)](#)

Other adapters: [cs_column_map\(\)](#), [cs_detect_format\(\)](#), [cs_read\(\)](#)

Examples

```
cs_formats()
```

cs_from_spe

Convert a SpatialExperiment back to cellspec

Description

Reconstructs the cellspec components stored by [cs_as_spe\(\)](#).

Usage

```
cs_from_spe(spe)
```

Arguments

spe A SpatialExperiment created by [cs_as_spe\(\)](#).

Value

A validated cellspec object.

See Also

Other interoperability: [cs_as_anndata\(\)](#), [cs_as_spe\(\)](#)

Examples

```
if (requireNamespace("SpatialExperiment", quietly = TRUE)) {
  x <- cs_example()
  policy <- cs_signal_policy(cs_markers(x))
  identical(cs_cells(x), cs_cells(cs_from_spe(cs_as_spe(x, policy))))
}
```

cs_marker_map

Create a marker-name mapping

Description

Builds a two-column mapping from source marker names to canonical marker names. Source names must be unique; several source names may map to the same canonical name when they represent the same marker across files.

Usage

```
cs_marker_map(from, to)
```

Arguments

from	Character vector of source marker names.
to	Character vector of canonical marker names, the same length as from.

Value

A two-column data frame with character columns from and to.

See Also

[cs_signal_policy\(\)](#)

Other signals: [cs_read_signal_policy\(\)](#), [cs_signal_matrix\(\)](#), [cs_signal_policy\(\)](#), [cs_write_signal_policy\(\)](#)

Examples

```
cs_marker_map(c("PanCK", "CD68"), c("Pan-Cytokeratin", "CD68"))
```

cs_merge_tiles	<i>Merge tiled cellspec objects using centroid ownership</i>
----------------	--

Description

Every centroid is assigned to the half open outer box containing it. A centroid found in more than one box is an error. Cells in a tile whose centroid is owned by another tile, or by no tile, are treated as halo cells and dropped.

Usage

```
cs_merge_tiles(tiles, tile_bounds, ownership = "centroid", tolerance = 1e-09)
```

Arguments

tiles	A named or positional list of cellspec tile objects. Names, when supplied, must equal tile_bounds\$tile_id.
tile_bounds	A data frame with tile_id, xmin, xmax, ymin and ymax in micrometres.
ownership	Ownership method. The only supported value is "centroid".
tolerance	Non-negative tolerance used to snap coordinates close to a box boundary before applying its half open comparisons.

Value

A merged cellspec object.

See Also

Other object: [cs_accessors](#), [cs_bind\(\)](#), [cs_example\(\)](#), [cs_features\(\)](#), [cs_new\(\)](#), [is_cellspec\(\)](#)

Examples

```
tile_bounds <- data.frame(  
  tile_id = "tile-1", xmin = 0, xmax = 10000, ymin = 0, ymax = 10000  
)  
cs_merge_tiles(list(cs_example()), tile_bounds)
```

cs_new

*Create a cellspec object***Description**

Builds a cellspec object from its components and checks it against the structure rules of the specification. Tool exports are normally read with `cs_read()`; use `cs_new()` when you already hold the pieces in memory, for example in another package that produces cell tables.

Inputs are normalised before validation: any data frame subclass becomes a plain `data.frame` without row names, factors become character vectors, identifier columns (`cell_id`, `image_id`, `sample_id`, `subject_id`) are converted to character, whole-number doubles in integer columns become integers, and `NaN/Inf` in measurements become `NA` (counted in `provenance$counts$nonfinite_converted`). A missing `sample_id` column in `cells` is filled from `images`.

Usage

```
cs_new(
  cells,
  measurements = NULL,
  dictionary = NULL,
  images,
  channels = NULL,
  provenance = NULL,
  adjacency = NULL
)
```

Arguments

<code>cells</code>	A data frame with one row per cell and at least the columns <code>cell_id</code> , <code>image_id</code> , <code>sample_id</code> (or a <code>sample_id</code> in <code>images</code>), <code>x</code> and <code>y</code> (centroids in micrometres).
<code>measurements</code>	A numeric matrix (or data frame of numeric columns) with one row per cell, in the row order of <code>cells</code> , and column names equal to <code>dictionary\$feature_id</code> . <code>NULL</code> means no features.
<code>dictionary</code>	A data frame describing each measurement column (see cs_vocabulary()). <code>NULL</code> is allowed only when there are no features.
<code>images</code>	A data frame with one row per image and at least <code>image_id</code> , <code>sample_id</code> and <code>pixel_size</code> .
<code>channels</code>	A data frame with one row per image and channel (<code>image_id</code> , <code>channel_index</code> , <code>channel_name</code> , <code>marker</code>), or <code>NULL</code> when no intensity features exist.
<code>provenance</code>	A list describing the origin of the data. Missing fields are filled with defaults; <code>NULL</code> records the object as built by <code>cs_new()</code> .
<code>adjacency</code>	Optional data frame of cell-cell contacts (<code>image_id</code> , <code>cell_id_a</code> , <code>cell_id_b</code> , <code>shared_boundary</code> , <code>method</code>).

Value

A cellspec object: a list of class "cellspec" with elements spec_version (string), cells (data frame), measurements (double matrix), dictionary, images, channels (data frames), provenance (list) and adjacency (data frame or NULL).

See Also

vignette("specification", package = "cellspecR") for every column and rule; [cs_validate\(\)](#) for the checks applied.

Other object: [cs_accessors](#), [cs_bind\(\)](#), [cs_example\(\)](#), [cs_features\(\)](#), [cs_merge_tiles\(\)](#), [is_cellspec\(\)](#)

Examples

```
cells <- data.frame(
  cell_id = c("1", "2", "3"),
  image_id = "img1",
  sample_id = "s1",
  x = c(10, 20, 30),
  y = c(5, 15, 25),
  area = c(50, 62.5, 48)
)
measurements <- cbind(
  "cell:CD3e:mean" = c(120, 8, 95),
  "nucleus:FOXP3:mean" = c(3, 1, 40)
)
dictionary <- data.frame(
  feature_id = colnames(measurements),
  kind = "intensity",
  marker = c("CD3e", "FOXP3"),
  compartment = c("cell", "nucleus"),
  statistic = "mean",
  unit = "a.u.",
  source_name = c("Cell: CD3e: Mean", "Nucleus: FOXP3: Mean")
)
images <- data.frame(image_id = "img1", sample_id = "s1", pixel_size = 0.5)
channels <- data.frame(
  image_id = "img1",
  channel_index = 1:3,
  channel_name = c("DAPI", "CD3e", "FOXP3"),
  marker = c("DAPI", "CD3e", "FOXP3")
)
x <- cs_new(cells, measurements, dictionary, images, channels)
x
```

Description

Reads a delimited cell table using a format-specific adapter or an explicit `cs_column_map()`. See `cs_formats()` for supported formats.

The segmantr adapter expects the one-based row and column centroid means returned by `segmentR::sg_extract_features`. It subtracts 0.5 before converting to micrometres and scales pixel area by `pixel_size^2`. Coordinates refer to the supplied image: cropped or downsampled inputs require an explicit external frame transform before combining with a full-resolution image. This adapter assumes square pixels. Shape measurements retain source units in the feature dictionary.

Usage

```
cs_read(
  path,
  format = "auto",
  pixel_size = NULL,
  image_id = NULL,
  sample_id = NULL,
  marker_map = NULL,
  column_map = NULL,
  tile_bounds = NULL,
  verify_done = TRUE,
  keep_other = TRUE,
  keep_paths = FALSE,
  quiet = FALSE
)
```

Arguments

<code>path</code>	One delimited table file.
<code>format</code>	Reader format. Use "table" for this adapter or "auto" to request conservative detection.
<code>pixel_size</code>	Micrometres per pixel. Required for pixel coordinates; micrometre-coordinate tables use 1 when no calibration is supplied.
<code>image_id</code>	Optional scalar image identifier overriding the mapped column; otherwise the mapped column or file basename is used.
<code>sample_id</code>	Optional scalar sample identifier overriding the mapped column; otherwise the mapped column or image identifier is used.
<code>marker_map</code>	Reserved for future adapters; not used by the table adapter.
<code>column_map</code>	A validated object returned by <code>cs_column_map()</code> .
<code>tile_bounds</code>	Optional tile-bound data frame for <code>format = "qupath_tiled"</code> , with <code>tile_id</code> , <code>xmin</code> , <code>xmax</code> , <code>ymin</code> and <code>ymax</code> in micrometres.
<code>verify_done</code>	Whether tiled readers should verify optional <code>.done.tsv</code> metadata against each source file.
<code>keep_other</code>	Whether unmapped numeric columns become other measurements and other unmapped columns are retained in cells. Every other feature, whether kept

here or mapped with `kind = "other"` in `column_map` (also when `keep_other = FALSE`), is identified as `other:` followed by its source name in lower case, with each run of characters other than `a-z` and `0-9` replaced by `_` and leading or trailing `_` removed (value if nothing is left). Identifiers that clash are made unique by `make.unique()` with `sep = "_"`, mapped measurements first in the order given, then unmapped columns in table order; a suffix that is already taken is skipped, so columns `a b`, `a_b` and `a_b_1` become `other:a_b`, `other:a_b_2` and `other:a_b_1`.

<code>keep_paths</code>	Whether the full input path may be recorded in provenance.
<code>quiet</code>	Reserved for consistency with future readers; suppresses the informational warning about dropped columns when <code>keep_other = FALSE</code> .

Value

A validated `cellspec` object.

Compressed input

Input files whose names end in `.gz` are decompressed in R and checked against their gzip trailer before they are parsed, for every format. A truncated or corrupt file, also one zero-filled back to its original size, raises an error of class `cellspec_error_format`. A BGZF file (written by `bgzip`) is checked block by block and must end with its end-of-file block. Other files made of several gzip members, such as concatenated `.gz` files, can be checked only in their last member; they are read with a warning of class `cellspec_warning_integrity`.

See Also

`cs_formats()`, `cs_detect_format()`, `cs_column_map()`

Other adapters: `cs_column_map()`, `cs_detect_format()`, `cs_formats()`

Examples

```
path <- tempfile(fileext = ".csv")
utils::write.csv(data.frame(id = c("c1", "c2"), x = 1:2, y = 3:4),
                 path, row.names = FALSE)
map <- cs_column_map("id", "x", "y")
x <- cs_read(path, format = "table", column_map = map)
dim(x)
```

`cs_read_cellspec`

Read a canonical `cellspec` directory

Description

Reads the sidecar and cell table written by `cs_write()`, verifies the manifest and DONE marker by default, and returns a validated object. DONE is required even when verification is explicitly disabled.

Usage

```
cs_read_cellspec(dir, verify = TRUE)
```

Arguments

dir	A canonical cellspec directory.
verify	Whether to verify all manifest digests before reading data.

Value

A validated cellspec object.

Compressed input

Tables written with `cs_write(format = "tsv.gz")` are decompressed in R and checked against their gzip trailer, also when `verify = FALSE`. A truncated or corrupt file, also one zero-filled back to its original size, raises an error of class `cellspec_error_format`. A BGZF file (written by `bgzip`) is checked block by block and must end with its end-of-file block. Other files made of several gzip members can be checked only in their last member; they are read with a warning of class `cellspec_warning_integrity`.

See Also

`cs_write()`, `cs_verify()`
 Other canonical-format: [cs_verify\(\)](#), [cs_write\(\)](#)

Examples

```
path <- tempfile("cellspec-")
cs_write(cs_example(), path)
y <- cs_read_cellspec(path)
identical(cs_cells(y), cs_cells(cs_example()))
```

cs_read_npy

Read and write NumPy array files

Description**[Experimental]**

`cs_read_npy()` reads one .npy file and `cs_read_npz()` reads the arrays of an .npz archive, both without Python. Cell tables exported from Python pipelines (centroids, labels, per-cell flags) can so be brought into `cs_new()` directly. `cs_write_npy()` and `cs_write_npz()` write the same formats, for example to hand results back or to build test fixtures.

Supported data types are booleans, signed and unsigned integers, 32- and 64-bit floats, 128-bit complex numbers, fixed-width Unicode (U) and byte (S) strings, in either byte order and in C or Fortran order. Object arrays (O, stored with Python's pickle) and structured data types are refused. 64-bit and unsigned 32-bit integers are returned as doubles; a value that a double cannot hold exactly is an error.

Usage

```

cs_read_npy(path)

cs_read_npz(path, arrays = NULL)

cs_write_npy(x, path)

cs_write_npz(x, path)

```

Arguments

path	Path to a .npy file (cs_read_npy(), cs_write_npy()) or a .npz archive (cs_read_npz(), cs_write_npz()).
arrays	Names of the arrays to read from the archive, without the .npz suffix. NULL reads all arrays.
x	For cs_write_npy(): a logical, integer, double or character vector, matrix or array. For cs_write_npz(): a named list of such objects.

Details

Arrays keep their logical shape: element $[i, j]$ in R is element $[i - 1, j - 1]$ in NumPy, whatever the storage order. Trailing NUL characters of fixed-width strings are dropped, as NumPy does.

cs_write_npz() stores the members uncompressed (as `numpy.savez()` does) and needs no external zip program. Character data are written as little-endian U strings, doubles as <f8, integers as <i4 and logical values as |b1; missing values are not allowed.

Value

cs_read_npy() returns a vector for one-dimensional arrays, a matrix or array with the same shape otherwise, and a length-one vector for zero-dimensional arrays. cs_read_npz() returns a named list of these, in archive order. The writers return path invisibly.

Examples

```

path <- tempfile(fileext = ".npz")
cs_write_npz(list(x_um = c(1.5, 2.5), label = c("Tumour", "Treg")), path)
cs_read_npz(path)
cs_read_npz(path, arrays = "label")$label

```

cs_read_signal_policy *Read a signal policy from JSON or CSV*

Description

Reads and validates a policy written by `cs_write_signal_policy()`. The format is inferred from the .json or .csv extension unless format is supplied explicitly.

Usage

```
cs_read_signal_policy(path, format = NULL)
```

Arguments

`path` Source file path.

`format` Either "json" or "csv"; by default inferred from path.

Value

A validated data frame with class `cs_signal_policy`.

See Also

[cs_write_signal_policy\(\)](#)

Other signals: [cs_marker_map\(\)](#), [cs_signal_matrix\(\)](#), [cs_signal_policy\(\)](#), [cs_write_signal_policy\(\)](#)

Examples

```
path <- tempfile(fileext = ".json")
cs_write_signal_policy(cs_signal_policy("CD3e"), path)
cs_read_signal_policy(path)
```

cs_signal_matrix	<i>Select one signal per cell and marker</i>
------------------	--

Description

Applies a `cs_signal_policy` to the intensity measurements in a `cellspec` object. A preferred finite value is used when its declared domain accepts it; otherwise the configured fallback is tested using its own domain. Default domains require values at least `min_value`. The "finite" domain accepts negative finite values. Non-finite values remain unavailable in both.

Usage

```
cs_signal_matrix(x, policy, image_id = NULL)
```

Arguments

`x` A `cellspec` object.

`policy` A `cs_signal_policy` created by [cs_signal_policy\(\)](#).

`image_id` Optional image identifier. When supplied, only cells from that image are returned.

Value

A list with `signal`, a cells by markers numeric matrix, `source`, a character matrix describing the selected source, and `policy`, the validated policy. Source values are "`<compartment>:<statistic>`", "`fallback:<compartment>:<statistic>`" or "`unavailable`".

See Also

`cs_signal_policy()`

Other signals: `cs_marker_map()`, `cs_read_signal_policy()`, `cs_signal_policy()`, `cs_write_signal_policy()`

Examples

```
x <- cs_example()
policy <- cs_signal_policy(cs_markers(x))
selected <- cs_signal_matrix(x, policy)
selected$signal[, 1:2]
```

cs_signal_policy	Create a signal selection policy
------------------	----------------------------------

Description

Creates one validated row per marker. Signal policies make the selected compartment, statistic, fallback and minimum usable value explicit.

Usage

```
cs_signal_policy(
  marker,
  compartment = "cell",
  statistic = "mean",
  fallback_compartment = NA_character_,
  min_value = 0,
  preferred_domain = "minimum",
  fallback_domain = "minimum"
)
```

Arguments

marker	Character vector of unique marker names.
compartment	Preferred compartment for each marker. A single value is recycled across marker.
statistic	Intensity statistic for each marker. A single value is recycled across marker.
fallback_compartment	Optional fallback compartment for each marker; NA disables fallback. A single value is recycled across marker.

`min_value` Minimum usable value for each marker. Values below this threshold are unavailable. A single value is recycled across marker.

`preferred_domain, fallback_domain` Usability domain for each branch: "minimum" requires a finite value at least `min_value`; "finite" accepts any finite value, including negatives, without applying the minimum. A plain character scalar is recycled across markers.

Value

A data frame with class `cs_signal_policy` and columns `marker`, `compartment`, `statistic`, `fallback_compartment` and `min_value`. Non-default domains append `preferred_domain` and `fallback_domain`. Default policies retain the original five-column representation.

See Also

[cs_signal_matrix\(\)](#), [cs_marker_map\(\)](#)

Other signals: [cs_marker_map\(\)](#), [cs_read_signal_policy\(\)](#), [cs_signal_matrix\(\)](#), [cs_write_signal_policy\(\)](#)

Examples

```
cs_signal_policy(
  marker = c("PanCK", "FOXP3"),
  compartment = c("cytoplasm", "nucleus"),
  fallback_compartment = "cell"
)
```

`cs_spec_version`

Version of the cellspec specification

Description

Returns the version of the cellspec format that this build of cellspecR writes and validates. Readers of major version 1 accept every 1.y file whose minor version is not newer than their own.

Usage

```
cs_spec_version()
```

Value

A single string in semantic versioning form, for example "1.0.0".

See Also

[vignette\("specification", package = "cellspecR"\)](#)

Other specification: [cs_vocabulary\(\)](#)

Examples

```
cs_spec_version()
```

 cs_validate

Validate a cellspec object

Description

cs_validate() checks an object against the cellspec specification and returns a report with one row per check. cs_assert_valid() runs the same checks and aborts when any check fails.

level = "structure" runs the MUST rules of the specification: required columns and types, unique keys, finite centroids, dictionary vocabulary, alignment of measurements, channels and optional adjacency. A failing structure check means other packages cannot rely on the object.

level = "semantic" adds plausibility checks that produce warnings, not failures: centroids outside the image, implausible cell areas, negative intensities, features with many missing values, markers without usable signal ([cs_feature_support\(\)](#)) and duplicated centroids, which usually indicate cells counted twice when tiles were merged.

Usage

```
cs_validate(
  x,
  level = c("structure", "semantic"),
  area_range = c(1, 5000),
  na_max = 0.01
)

cs_assert_valid(x, level = "structure", call = rlang::caller_env())
```

Arguments

x	A cellspec object.
level	"structure" or "semantic" (structure plus semantic checks).
area_range	Numeric vector of length 2: plausible whole-cell area in square micrometres for the area_range check.
na_max	Largest acceptable fraction of missing values per feature for the feature_na_fraction check.
call	The execution environment used in error messages; internal callers pass their own.

Value

`cs_validate()` returns a data frame of class `cellspec_validation` with one row per check and the columns `check` (character), `level` ("structure" or "semantic"), `status` ("pass", "warn", "fail", or "skip" when a prerequisite check failed), `n` (integer number of offending rows, features or values) and `message` (character, naming columns or features and up to five examples). Rows are in catalogue order; `print()` shows failures first.

`cs_assert_valid()` returns `x` invisibly, or aborts with an error of class `cellspec_error_invalid` whose `report` field holds the report.

See Also

`vignette("specification", package = "cellspecR")`

Other validation: [cs_feature_support\(\)](#)

Examples

```
x <- cs_example()
cs_validate(x)
report <- cs_validate(x, level = "semantic")
report[report$status != "pass", ]

# A broken object fails with a classed error
bad <- x
bad$cells$x[1] <- NA
try(cs_assert_valid(bad))
```

cs_verify

Verify a canonical cellspec directory

Description

Recomputes every digest in `MANIFEST.sha256` and checks the `DONE` marker against the manifest digest. The returned report has one row per declared file plus `DONE`; it does not hide a failed digest behind an error.

Usage

```
cs_verify(dir)
```

Arguments

`dir` A cellspec directory written by `cs_write()`.

Value

A data frame with character columns `file`, `expected` and `observed`, and logical column `ok`. `ok` is `FALSE` for a missing, modified or unexpected file, or for an invalid `DONE` marker.

See Also

`cs_read_cellspec()`

Other canonical-format: `cs_read_cellspec()`, `cs_write()`

Examples

```
path <- tempfile("cellspec-")
cs_write(cs_example(), path)
cs_verify(path)
```

cs_vocabulary

Columns and controlled vocabularies of the cellspec format

Description

Returns the specification tables as data: the columns each component may carry, and the allowed values of every controlled vocabulary. The validator (`cs_validate()`) enforces exactly these tables, so other packages can use them to build compatible objects.

Usage

```
cs_vocabulary(what = c("columns", "vocabularies", "checks"))
```

Arguments

`what` Which table to return: "columns", "vocabularies" or "checks".

Value

For `what = "columns"`, a data frame with one row per specified column and the character columns component ("cells", "dictionary", "images", "channels", "adjacency"), column, type (R storage type), requirement ("MUST", "SHOULD", "MAY") and description. For `what = "vocabularies"`, a data frame with the character columns vocabulary, value and description. For `what = "checks"`, the validation catalogue used by `cs_validate()`: character columns check, level ("structure" or "semantic") and description.

See Also

`vignette("specification", package = "cellspecR")`

Other specification: `cs_spec_version()`

Examples

```
cols <- cs_vocabulary("columns")
cols[cols$component == "cells", c("column", "type", "requirement")]

voc <- cs_vocabulary("vocabularies")
voc$value[voc$vocabulary == "compartment"]

head(cs_vocabulary("checks"))
```

cs_write	<i>Write a cellspec directory</i>
----------	-----------------------------------

Description

Writes a canonical cellspec directory in a neighbouring staging directory, then renames it into place after the data, sidecar, manifest and DONE marker are complete. Parquet is the default because it preserves doubles exactly; the tsv.gz option is available for interoperable text output.

Usage

```
cs_write(x, dir, format = c("parquet", "tsv.gz"), overwrite = FALSE)
```

Arguments

x	A valid cellspec object.
dir	Destination directory. Its parent is created when needed.
format	Storage format: parquet or tsv.gz.
overwrite	Whether an existing destination may be replaced. Only an empty directory or earlier cs_write() output is replaced: a directory whose cellspec.json carries the cellspec format marker and which holds nothing but the files cs_write() creates. Any other directory, for example one with an added file, is left untouched and an error naming the unexpected entries is raised.

Value

x, invisibly.

See Also

cs_read_cellspec(), cs_verify()

Other canonical-format: [cs_read_cellspec\(\)](#), [cs_verify\(\)](#)

Examples

```
path <- tempfile("cellspec-")
cs_write(cs_example(), path)
```

cs_write_signal_policy	<i>Write a signal policy as JSON or CSV</i>
------------------------	---

Description

Writes a validated signal policy. The format is inferred from the .json or .csv extension unless format is supplied explicitly.

Usage

```
cs_write_signal_policy(policy, path, format = NULL)
```

Arguments

- policy A cs_signal_policy object.
- path Destination file path.
- format Either "json" or "csv"; by default inferred from path.

Value

policy, invisibly.

See Also

[cs_read_signal_policy\(\)](#)
Other signals: [cs_marker_map\(\)](#), [cs_read_signal_policy\(\)](#), [cs_signal_matrix\(\)](#), [cs_signal_policy\(\)](#)

Examples

```
path <- tempfile(fileext = ".json")
cs_write_signal_policy(cs_signal_policy("CD3e"), path)
```

is_cellspec	<i>Test for a cellspec object</i>
-------------	-----------------------------------

Description

Test for a cellspec object

Usage

```
is_cellspec(x)
```

Arguments

x Any R object.

Value

TRUE if x inherits from "cellspec", otherwise FALSE. The object is not validated; use `cs_validate()` for that.

See Also

Other object: `cs_accessors`, `cs_bind()`, `cs_example()`, `cs_features()`, `cs_merge_tiles()`, `cs_new()`

Examples

```
is_cellspec(cs_example())
is_cellspec(data.frame())
```

plot.cellspec

Plot cellspec quality-control views

Description

Draws a centroid map, marker distribution, feature-support heatmap, cell area distribution, or an overview of up to four panels. The overview leaves out the area panel when the cells table has no area column and the marker panel when there are no intensity features. Large objects are subsampled deterministically for plotting.

Usage

```
## S3 method for class 'cellspec'
plot(
  x,
  type = c("overview", "map", "marker", "support", "area"),
  colour_by = NULL,
  marker = NULL,
  policy = NULL,
  max_points = 50000L,
  seed = 1L,
  ...
)
```

Arguments

x	A cellspec object.
type	Plot type: "overview", "map", "marker", "support" or "area".
colour_by	A cells or measurement column used by the map.
marker	Marker used by the marker plot.
policy	Optional signal policy for the marker plot.
max_points	Maximum number of cells used by a plot.
seed	Seed used for deterministic subsampling.
...	Reserved for future plot-specific parameters.

Value

A ggplot object, or a patchwork object for type = "overview".

See Also

Other methods: [\[.cellspec\(\)\]](#), [as.data.frame.cellspec\(\)](#), [cellspec-methods](#)

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {  
  plot(cs_example(), type = "map")  
}
```

Index

- * **adapters**
 - cs_column_map, 10
 - cs_detect_format, 11
 - cs_formats, 15
 - cs_read, 19
- * **app**
 - cs_app, 7
- * **canonical-format**
 - cs_read_cellspec, 21
 - cs_verify, 28
 - cs_write, 30
- * **interoperability**
 - cs_as_anndata, 8
 - cs_as_spe, 9
 - cs_from_spe, 15
- * **io**
 - cs_read_npy, 22
- * **methods**
 - [.cellspec, 3
 - as.data.frame.cellspec, 4
 - cellspec-methods, 5
 - plot.cellspec, 32
- * **object**
 - cs_accessors, 6
 - cs_bind, 10
 - cs_example, 12
 - cs_features, 14
 - cs_merge_tiles, 17
 - cs_new, 18
 - is_cellspec, 31
- * **signals**
 - cs_marker_map, 16
 - cs_read_signal_policy, 23
 - cs_signal_matrix, 24
 - cs_signal_policy, 25
 - cs_write_signal_policy, 31
- * **specification**
 - cs_spec_version, 26
 - cs_vocabulary, 29
- * **validation**
 - cs_feature_support, 13
 - cs_validate, 27
 - [.cellspec, 3
 - [.cellspec(), 4, 5, 33

 - as.data.frame.cellspec, 4
 - as.data.frame.cellspec(), 3, 5, 33
 - as_tibble.cellspec
 - (as.data.frame.cellspec), 4

 - cellspec-methods, 5
 - cs_accessors, 6, 10, 12, 14, 17, 19, 32
 - cs_adjacency (cs_accessors), 6
 - cs_app, 7
 - cs_as_anndata, 8
 - cs_as_anndata(), 9, 16
 - cs_as_spe, 9
 - cs_as_spe(), 8, 15, 16
 - cs_assert_valid (cs_validate), 27
 - cs_bind, 10
 - cs_bind(), 6, 12, 14, 17, 19, 32
 - cs_cells (cs_accessors), 6
 - cs_channels (cs_accessors), 6
 - cs_column_map, 10
 - cs_column_map(), 12, 15, 20, 21
 - cs_detect_format, 11
 - cs_detect_format(), 11, 15, 21
 - cs_dictionary (cs_accessors), 6
 - cs_example, 12
 - cs_example(), 6, 10, 14, 17, 19, 32
 - cs_feature_support, 13
 - cs_feature_support(), 27, 28
 - cs_features, 14
 - cs_features(), 6, 10, 12, 17, 19, 32
 - cs_formats, 15
 - cs_formats(), 11, 12, 20, 21
 - cs_from_spe, 15
 - cs_from_spe(), 8, 9
 - cs_images (cs_accessors), 6

`cs_marker_map`, 16
`cs_marker_map()`, 24–26, 31
`cs_markers (cs_features)`, 14
`cs_measurements (cs_accessors)`, 6
`cs_merge_tiles`, 17
`cs_merge_tiles()`, 6, 10, 12, 14, 19, 32
`cs_new`, 18
`cs_new()`, 6, 10, 12, 14, 17, 32
`cs_provenance (cs_accessors)`, 6
`cs_read`, 19
`cs_read()`, 10–12, 15
`cs_read_cellspec`, 21
`cs_read_cellspec()`, 29, 30
`cs_read_npy`, 22
`cs_read_npz (cs_read_npy)`, 22
`cs_read_signal_policy`, 23
`cs_read_signal_policy()`, 16, 25, 26, 31
`cs_signal_matrix`, 24
`cs_signal_matrix()`, 16, 24, 26, 31
`cs_signal_policy`, 25
`cs_signal_policy()`, 16, 24, 25, 31
`cs_spec_version`, 26
`cs_spec_version()`, 29
`cs_validate`, 27
`cs_validate()`, 13, 19, 29, 32
`cs_verify`, 28
`cs_verify()`, 22, 30
`cs_vocabulary`, 29
`cs_vocabulary()`, 18, 26
`cs_write`, 30
`cs_write()`, 22, 29
`cs_write_npy (cs_read_npy)`, 22
`cs_write_npz (cs_read_npy)`, 22
`cs_write_signal_policy`, 31
`cs_write_signal_policy()`, 16, 23–26

`dim.cellspec (cellspec-methods)`, 5

`is_cellspec`, 31
`is_cellspec()`, 6, 10, 12, 14, 17, 19

`make.unique()`, 21

`plot.cellspec`, 32
`plot.cellspec()`, 3–5
`print.cellspec (cellspec-methods)`, 5
`print.cellspec_summary`
 (`cellspec-methods`), 5

`summary.cellspec (cellspec-methods)`, 5