Package: cellreportR (via r-universe)

November 3, 2026

Title Cell Culture Microscopy Assay Analysis and Reporting

Version 0.2.1.9000

Description Analysis and reporting pipeline for cell culture assays read out by microscopy, starting from segmented single-cell exports such as those written by 'segmantR', 'CellProfiler' or 'QuPath'. Recovers the experimental design from the directory tree and the file naming convention, resolves acquisitions into the analysis unit of replication, and standardizes every cell against the control condition of its own batch. Applies threshold and biological quality control gates, then estimates effect sizes with confidence intervals at both unit and cell level so that the cost of treating cells as replicates is visible rather than assumed. Sample sizes for a follow-up study are solved from the confidence bound nearer the null instead of the observed point estimate, after Cohen (1988, ISBN:9780805802832). Figures, machine-readable tables and a generated file of every quoted number are emitted from the same objects, and an interactive 'shiny' application provides guided analysis and report export.

License MIT + file LICENSE

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

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

Depends R (>= 4.1.0)

Imports cli, digest, dplyr, ggplot2, jsonlite, pROC, readr, readxl, rlang, scales, shiny, stats, tibble, tidyr, tools, utils

Suggests bslib, DT, flowCore, knitr, plotly, rmarkdown, shinytest2, shinyWidgets, sva, svglite, testthat (>= 3.0.0), withr, writexl

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

Config/Needs/website CTTIR/themakR

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libx11-dev zlib1g-dev

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

Date/Publication 2026-11-03 11:31:12 UTC

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

RemoteRef HEAD

RemoteSha 987ac519f6364e243d8ccb0972d07a0f44adaa5d

Contents

cr_apply_gate . . . 5
cr_assign_units . . . 6
cr_auc . . . 7
cr_background_subtract . . . 8
cr_balance_cells . . . 9
cr_batch_key . . . 10
cr_batch_reference . . . 11
cr_blocked_effect . . . 13
cr_build_experiment . . . 14
cr_centroid_overlap . . . 16
cr_channels . . . 17
cr_classify_result . . . 18
cr_column_map . . . 19
cr_compare_levels . . . 20
cr_compute_metrics . . . 21
cr_confusion_matrix . . . 22
cr_conservative_effect . . . 23
cr_correct_batch . . . 23
cr_dataset . . . 25
cr_design . . . 26
cr_dose_response . . . 28
cr_effect_grid . . . 29
cr_effect_size . . . 31
cr_enumerate . . . 32
cr_example_design . . . 33
cr_example_experiment . . . 34
cr_example_exports . . . 36
cr_example_files . . . 37
cr_example_path . . . 38
cr_example_screen . . . 39
cr_exclude_small . . . 41
cr_export_plots . . . 42
cr_export_report_audit . . . 43

cr_export_report_spec . . . 44
cr_export_results . . . 44
cr_export_tables . . . 45
cr_extract_markers . . . 46
cr_figure_caption . . . 47
cr_figure_size . . . 48
cr_filename_grammar . . . 48
cr_filter_cells . . . 50
cr_finalize_result . . . 51
cr_fold_change . . . 51
cr_format_interval . . . 53
cr_format_number . . . 53
cr_format_percent . . . 54
cr_format_pvalue . . . 54
cr_ic50 . . . 55
cr_import_report_spec . . . 55
cr_interpretation_lookup . . . 56
cr_lab_report . . . 56
cr_lab_report_from_evidence . . . 57
cr_linetypes . . . 58
cr_logistic . . . 58
cr_macro_name . . . 59
cr_macros . . . 60
cr_macros_from . . . 61
cr_marker_rules . . . 62
cr_merge_experiments . . . 64
cr_merge_rules . . . 65
cr_n_cells . . . 66
cr_normalize . . . 67
cr_palette . . . 68
cr_parse_paths . . . 69
cr_path_spec . . . 71
cr_plot_comparison . . . 72
cr_plot_dose_response . . . 73
cr_plot_effect_sizes . . . 73
cr_plot_estimates . . . 74
cr_plot_foldchange . . . 75
cr_plot_forest . . . 76
cr_plot_heatmap . . . 77
cr_plot_histogram . . . 78
cr_plot_intensity . . . 79
cr_plot_plate . . . 80
cr_plot_qc . . . 81
cr_plot_qc_gate . . . 81
cr_plot_qc_summary . . . 83
cr_plot_result_position . . . 83
cr_plot_roc . . . 84
cr_plot_sample_size . . . 85

cr_plot_scatter . . . 86
cr_plot_screen . . . 87
cr_plot_spatial . . . 89
cr_plot_specificity . . . 90
cr_plot_style . . . 91
cr_plot_timeline . . . 92
cr_power . . . 93
cr_power_analysis . . . 94
cr_power_grid . . . 95
cr_print_report_spec . . . 96
cr_qc_doublets . . . 97
cr_qc_filter . . . 98
cr_qc_gate . . . 99
cr_qc_gate_impact . . . 101
cr_qc_intensity . . . 102
cr_qc_manual . . . 103
cr_qc_report . . . 104
cr_qc_summary . . . 105
cr_read_cellprofiler . . . 105
cr_read_cells . . . 106
cr_read_design . . . 107
cr_read_export . . . 108
cr_read_exports . . . 109
cr_read_qupath . . . 111
cr_read_segmantr . . . 112
cr_render_lab_report . . . 113
cr_render_report . . . 114
cr_report . . . 115
cr_report_data . . . 117
cr_report_display_data . . . 117
cr_report_evidence . . . 118
cr_report_hash . . . 119
cr_report_profile . . . 119
cr_report_provenance . . . 120
cr_report_qc . . . 121
cr_report_qc_from_log . . . 121
cr_report_review_app . . . 122
cr_report_spec . . . 123
cr_report_style . . . 124
cr_roc . . . 126
cr_rowcol_to_well . . . 126
cr_run_app . . . 127
cr_save_figure . . . 128
cr_save_plot . . . 129
cr_scale_colour . . . 130
cr_scale_diverging . . . 131
cr_scale_group . . . 131
cr_shapes . . . 132

cr_standardize_batch . . . 133
cr_summarize_wells . . . 135
cr_table_disposition . . . 136
cr_table_qc . . . 137
cr_tables . . . 138
cr_test . . . 139
cr_test_all . . . 140
cr_theme . . . 141
cr_unit_map . . . 142
cr_unit_variability . . . 143
cr_validate_experiment . . . 144
cr_validate_report_evidence . . . 145
cr_validate_report_qc . . . 146
cr_validate_report_spec . . . 146
cr_well_to_rowcol . . . 147
print.cr_experiment . . . 147
print.cr_qc_gate . . . 148
print.cr_report . . . 148
print.cr_result . . . 149
summary.cr_experiment . . . 149
summary.cr_report . . . 150
summary.cr_result . . . 150

Index **151**

`cr_apply_gate` *Apply a QC gate to an experiment*

Description

Removes the cells of every unit the gate excluded, records the step in the QC log and stores the gate on the experiment so the decision remains auditable after the data have been trimmed.

Usage

```
cr_apply_gate(experiment, gate, units = NULL, drop_disputed = FALSE)
```

Arguments

`experiment`	A `cr_experiment`.
`gate`	A `cr_qc_gate` from `cr_qc_gate()`.
`units`	Optional character vector of unit identifiers to remove instead of `gate$excluded`, for a reviewed exclusion list.
`drop_disputed`	Also drop the units whose verdict depends on which control centre is used. Default `FALSE`.

Value

A modified `cr_experiment`. The gate is stored in `metadata$qc_gate`.

See Also

`cr_qc_gate()`, `cr_qc_gate_impact()`, `cr_qc_report()`.

Other quality control: `cr_balance_cells()`, `cr_exclude_small()`, `cr_qc_doublets()`, `cr_qc_filter()`, `cr_qc_gate()`, `cr_qc_gate_impact()`, `cr_qc_intensity()`, `cr_qc_manual()`, `cr_qc_report()`, `cr_qc_summary()`, `print.cr_qc_gate()`

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 40)
gate <- cr_qc_gate(exp, "marker_1", "Untreated", batch_vars = "replicate")
exp2 <- cr_apply_gate(exp, gate)
cr_qc_summary(exp2)
```

---

cr_assign_units *Assign cells to analysis units*

Description

Derives the identifier of the analysis unit — the spatial unit that is the unit of replication — for every row of a cell table, merging the files that belong to the same unit according to a `cr_merge_rules()` set.

Usage

```
cr_assign_units(
  x,
  key_vars,
  replicate_var = "replicate",
  rules = cr_merge_rules(),
  id_col = "well_id",
  sep = "|",
  call = rlang::caller_env()
)
```

Arguments

`x`	A data frame of cells (or of files) carrying the design columns and a replicate index.
`key_vars`	Character vector of column names that, together with the replicate index, identify one unit.
`replicate_var`	Name of the replicate index column. Default `"replicate"`.
`rules`	A `cr_merge_rules()` object.

id_col	Name of the unit identifier column to add. Default "well_id".
sep	Separator used to build the identifier. Default "|".
call	The execution environment of the calling function. Used for error reporting; experts only.

Details

The unit identifier is the combination of key_vars and the merged replicate index. Files that share it are one unit; files that do not are separate replicates. Because a unit may be assembled from more than one file, cr_unit_map() should be used afterwards to see which units were merged and from how many files.

Value

x with two columns added: replicate_merged (the replicate index after merging) and the unit identifier named by id_col. The number of units is attached as the "n_units" attribute.

See Also

cr_unit_map(), cr_merge_rules(), cr_centroid_overlap().

Other import: cr_centroid_overlap(), cr_column_map(), cr_extract_markers(), cr_filename_grammar(), cr_marker_rules(), cr_merge_rules(), cr_parse_paths(), cr_path_spec(), cr_read_cellprofiler(), cr_read_cells(), cr_read_design(), cr_read_export(), cr_read_exports(), cr_read_qupath(), cr_read_segmantr(), cr_unit_map()

Examples

```
files <- tibble::tibble(
  compound = "CompoundA",
  plate = "Plate_1",
  mode = "treated",
  replicate = c("1", "1.1", "2", "2.2"),
  merge_unit = c(FALSE, TRUE, FALSE, FALSE),
  reacquisition = FALSE
)
units <- cr_assign_units(files, key_vars = c("compound", "plate", "mode"))
units[, c("replicate", "replicate_merged", "well_id")]
```

---

cr_auc *Compute AUC with confidence interval from a* cr_result

---

Description

Compute AUC with confidence interval from a cr_result

Usage

```
cr_auc(result, ci_method = c("delong", "bootstrap"), n_boot = 1000)
```

Arguments

result	A cr_result from cr_logistic().
ci_method	"delong" (default) or "bootstrap".
n_boot	Number of bootstrap resamples for "bootstrap".

Value

A tibble with auc, ci_low, ci_high, method.

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
res <- cr_logistic(exp, "marker_1", "CompoundA_high", "Untreated")
cr_auc(res)
```

---

cr_background_subtract

Subtract background from a channel

Description

Computes a per-well background estimate and subtracts it from every cell in that well. Negative values are clipped to zero.

Usage

```
cr_background_subtract(
  experiment,
  channel,
  method = c("percentile", "modal", "empty_wells"),
  q = 0.05,
  empty_wells = NULL
)
```

Arguments

experiment	A cr_experiment.
channel	Channel name.
method	Background estimator. "percentile" (default) uses a low percentile (default 5th), "modal" uses the mode of a kernel density estimate, and "empty_wells" uses the median intensity of wells listed in empty_wells.
q	Percentile for "percentile" method (0-1, default 0.05).
empty_wells	Character vector of well IDs for "empty_wells".

Value

A modified `cr_experiment`.

See Also

`cr_normalize()`, `cr_standardize_batch()`

Other normalization functions: `cr_normalize()`

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
cr_background_subtract(exp, channel = "marker_1")
```

---

cr_balance_cells *Balance the number of cells per analysis unit*

Description

Randomly subsamples each analysis unit to a common cell count. A unit acquired in several passes otherwise contributes several times as many cells as its single-pass neighbour and is silently over-weighted in every pooled quantity they share — including the control statistics of their own batch.

Usage

```
cr_balance_cells(experiment, unit = NULL, n_max = NULL, n = NULL, seed = NULL)
```

Arguments

`experiment`	A `cr_experiment`.
`unit`	Column in `cells` identifying the analysis unit. Defaults to the experiment's unit column, falling back to its spatial unit (`well` / `slide`).
`n_max`	Optional maximum number of cells per unit.
`n`	Optional exact number of cells per unit. Cannot be combined with `n_max`.
`seed`	Optional integer seed.

Details

Three modes:

`n_max` cap each unit at `n_max` cells, leaving smaller units untouched. This is the usual choice for large acquisitions.

`n` take exactly `n` cells per unit (units with fewer cells keep all of theirs).

neither take the smallest unit's cell count from every unit, which equalises the units exactly at the cost of discarding the most data.

Subsampling is random, so pass `seed` for a reproducible result. The RNG state of the calling session is saved and restored, so a seeded stream outside this function is never disturbed.

Value

A modified `cr_experiment`. Per-unit counts before and after are stored in `metadata$balance_cells`.

See Also

`cr_exclude_small()`, which should run first.

Other quality control: `cr_apply_gate()`, `cr_exclude_small()`, `cr_qc_doublets()`, `cr_qc_filter()`, `cr_qc_gate()`, `cr_qc_gate_impact()`, `cr_qc_intensity()`, `cr_qc_manual()`, `cr_qc_report()`, `cr_qc_summary()`, `print.cr_qc_gate()`

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 40)
exp2 <- cr_balance_cells(exp, n_max = 20, seed = 1)
max(exp2$metadata$balance_cells$n_after)

# equalise every unit at the smallest unit's count
exp3 <- cr_balance_cells(exp, seed = 1)
length(unique(exp3$metadata$balance_cells$n_after))
```

---

cr_batch_key *Construct a batch key*

Description

A batch is rarely identified by a single column. Once an assay spans several plates, acquisition runs or experiment days, the analytical batch is the *combination* of those columns, and any analysis that standardizes against a control has to respect that combination. `cr_batch_key()` collapses any number of design or cell columns into one character key that can be grouped on, joined on and reported.

Usage

```
cr_batch_key(x, batch_vars, sep = " | ", na_label = "<NA>")
```

Arguments

`x`	A `cr_experiment` or a data frame. For a `cr_experiment` the design columns are joined onto the cells first; a column present in both tables is taken from `cells`.
`batch_vars`	Character vector of column names that jointly define a batch, for example `c("compound", "run", "plate", "experiment", "interval")`.
`sep`	Separator placed between the parts of the key.
`na_label`	Label substituted for missing values.

Details

Missing values are replaced by `na_label` rather than propagated, so that two rows with an unrecorded plate are grouped together visibly instead of silently producing `NA` keys. If distinct tuples produce the same displayed key, the function refuses the collision; choose another separator or missing-value label.

Value

A character vector with one key per row of `x` (one key per cell when `x` is a `cr_experiment`).

See Also

`cr_batch_reference()`, `cr_standardize_batch()`

Other batch standardization functions: `cr_batch_reference()`, `cr_correct_batch()`, `cr_standardize_batch()`

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 10)
exp$design$plate <- rep(c("P1", "P2"), length.out = nrow(exp$design))
key <- cr_batch_key(exp, c("plate", "timepoint"))
head(unique(key))

# Works on a plain data frame too
cr_batch_key(data.frame(a = c("x", "y"), b = 1:2), c("a", "b"))
```

`cr_batch_reference` *Per-batch control reference statistics*

Description

Summarizes the control cells of every batch: how many there are, and their mean, median and standard deviation for one channel. This table is the reference that `cr_standardize_batch()` divides by, and it is worth inspecting on its own – a batch with very few control cells, or none at all, is visible here before any standardized value is computed.

Usage

```
cr_batch_reference(
  experiment,
  channel,
  control_level,
  batch_vars,
  control_var = "treatment",
  sd_floor = 1e-08
)
```

Arguments

experiment	A cr_experiment.
channel	Channel column to summarize. Must be numeric.
control_level	Value (or values) of control_var that mark the control cells of a batch.
batch_vars	Character vector of columns that jointly define a batch. See cr_batch_key().
control_var	Column holding the treatment assignment. Defaults to "treatment".
sd_floor	Lower bound for the control standard deviation. A batch whose control cells are identical would otherwise divide by zero.

Details

Both centres are reported deliberately. A right-skewed signal has mean > median, so a downstream gate that compares a well's median against a control *mean* is silently stricter than the rule it states. Keeping both lets each consumer compare like with like.

Value

A tibble with one row per batch and the columns

batch_vars The columns that define the batch.

batch_key The collapsed batch key.

n_cells Cells in the batch.

ctrl_n Control cells with a finite channel value.

ctrl_mean, ctrl_median, ctrl_sd Control statistics; NA when the batch has no control cells.

has_control Whether the batch can be standardized.

See Also

cr_standardize_batch(), cr_batch_key()

Other batch standardization functions: cr_batch_key(), cr_correct_batch(), cr_standardize_batch()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 20)
exp$design$plate <- rep(c("P1", "P2"), length.out = nrow(exp$design))
cr_batch_reference(exp, channel = "marker_1",
                   control_level = "Untreated", batch_vars = "plate")
```

`cr_blocked_effect` *Block-stratified sensitivity fit*

Description

Re-estimates each contrast *within* a blocking factor, typically the plate. Units on one plate share a preparation run, an imaging session and a single control denominator, so they are not independent and a pooled estimate carries the between-plate variance inside its standard deviation. Fitting `value ~ group + block` on unit-level data removes the additive block effect and shows whether the pooled headline estimate is conservative or inflated by pooling.

Usage

```
cr_blocked_effect(
  data,
  value,
  group_var,
  reference_level,
  comparison_levels = NULL,
  block_var,
  by = NULL,
  unit = NULL,
  conf_level = 0.95,
  min_blocks = 2
)
```

Arguments

`data`	A data frame with one row per unit (or per cell when `unit` is given), or a `cr_experiment`.
`value`	Name of the numeric response column.
`group_var`	Name of the column holding the compared groups.
`reference_level`	Reference level of `group_var`.
`comparison_levels`	Levels contrasted against the reference. `NULL` (default) uses every other level present.
`block_var`	Name of the blocking column, for example the plate.
`by`	Optional character vector of stratifying columns, for example the compound.
`unit`	Optional column identifying the unit of replication. When given, rows are averaged per unit before the model is fitted.
`conf_level`	Confidence level for the coefficient interval.
`min_blocks`	Minimum number of blocks that must carry data for a contrast to be fitted (default 2; with one block the model is the unblocked one).

Details

The reported d_within_block is the group coefficient divided by the residual standard deviation of the blocked fit, so it is on the same scale as Cohen's *d* from cr_effect_grid() but conditions on the block.

Value

A tibble with the by columns, contrast, n_units, n_blocks, shift_within_block (the group coefficient), ci_low, ci_high, resid_sd, d_within_block and p_value. Contrasts that could not be fitted are returned in the skipped attribute.

See Also

cr_effect_grid(), cr_unit_variability().

Other screen statistics: cr_unit_variability()

Examples

```
set.seed(1)
units <- data.frame(
  compound = rep(c("CompoundA", "CompoundB"), each = 16),
  arm = rep(rep(c("reference", "interval_short"), each = 8), 2),
  plate = rep(rep(c("P1", "P2"), each = 4), 8),
  log2_fc = stats::rnorm(32)
)
units$log2_fc <- units$log2_fc +
  ifelse(units$compound == "CompoundA" & units$arm == "interval_short",
         -1.2, 0) +
  ifelse(units$plate == "P2", 0.5, 0)
cr_blocked_effect(units, value = "log2_fc", group_var = "arm",
                  reference_level = "reference",
                  block_var = "plate", by = "compound")
```

---

cr_build_experiment *Build a* cr_experiment *object*

Description

Assembles a validated cr_experiment from its components. A cr_experiment is the central S3 object in cellreportR and holds per-cell measurements, experimental design, channel metadata, plate information, a QC log and arbitrary user metadata.

Usage

```
cr_build_experiment(
  cells,
  design = NULL,
  channels = NULL,
  plate_info = list(),
  metadata = list(),
  unit_var = NULL,
  batch_vars = NULL,
  provenance = NULL,
  set_aside = NULL,
  call = rlang::caller_env()
)
```

Arguments

`cells`	A data frame / tibble of per-cell measurements, or a `cr_dataset()`. Must contain a `cell_id` column and a spatial unit column (`well`, `slide`, `well_id` or `unit`, or the column named by `unit_var`).
`design`	A data frame / tibble that maps each spatial unit to treatment information, or a `cr_design()` object. Must contain the spatial unit column and a `treatment` column. Recommended columns: `dose`, `dose_unit`, `replicate`, `group`, `timepoint`. May be `NULL` when `cells` is a `cr_dataset` carrying a design.
`channels`	Optional tibble describing marker channels. Columns: `channel` (required), `role`, `target`, `fluorophore`. If `NULL`, channels are auto-detected from numeric columns in `cells` that are not recognised as morphology fields.
`plate_info`	Optional list with plate metadata (e.g. `format = "96"`, `microscope`, `date`, `operator`).
`metadata`	Optional list with arbitrary user metadata.
`unit_var`	Optional name of the analysis unit column.
`batch_vars`	Optional character vector of columns that together define a batch.
`provenance`	Optional per-file provenance table.
`set_aside`	Optional data frame or list of arms split out of the analysis pool.
`call`	The execution environment of the calling function. Used for error reporting; experts only.

Details

Three optional slots describe structure that a single design table cannot: `unit_var` names the analysis unit when it is neither `well` nor `slide` (a unit assembled from several files, for instance), `batch_vars` names the *combination* of columns that defines a batch, and `provenance` keeps the per-file record that lets any cell be traced back to its acquisition. `set_aside` holds arms that were split out of the analysis pool, such as a specificity control, so that they travel with the experiment instead of being lost.

Value

A `cr_experiment` object (an S3 list).

See Also

`cr_validate_experiment()`, `cr_dataset()`, `cr_design()`.

Other constructors: `cr_dataset()`, `cr_design()`, `cr_validate_experiment()`

Examples

```
cells <- tibble::tibble(
  cell_id = sprintf("c%03d", 1:6),
  well = rep(c("A01", "A02"), each = 3),
  area = c(120, 130, 125, 118, 122, 131),
  target_signal = c(10, 12, 11, 30, 33, 29)
)
design <- tibble::tibble(
  well = c("A01", "A02"),
  treatment = c("Vehicle", "CompoundA"),
  plate = "Plate_1"
)
exp <- cr_build_experiment(cells, design, batch_vars = "plate")
exp
```

---

cr_centroid_overlap *Centroid overlap between two candidate units*

Description

Answers the question a mergeable-looking replicate suffix raises: are these two files two passes over the same physical unit, or two different units? Two passes over one unit share their segmentation centroids; two different units share essentially none.

Usage

```
cr_centroid_overlap(
  x,
  unit_a,
  unit_b,
  coords = c("x", "y"),
  tol = 1,
  id_col = "well_id",
  call = rlang::caller_env()
)
```

Arguments

x	A cell table carrying a unit identifier and centroid columns.
unit_a, unit_b	The two unit identifiers to compare.
coords	Length-2 character vector naming the centroid columns. Default c("x", "y").
tol	Numeric matching tolerance in centroid units. Default 1.
id_col	Name of the unit identifier column. Default "well_id".
call	The execution environment of the calling function. Used for error reporting; experts only.

Value

A single number: the fraction of cells in the smaller unit that have a centroid within tol of a centroid in the other unit. The cell counts and the number of matches are attached as the "n_a", "n_b" and "n_matched" attributes.

See Also

cr_assign_units(), cr_merge_rules().

Other import: cr_assign_units(), cr_column_map(), cr_extract_markers(), cr_filename_grammar(), cr_marker_rules(), cr_merge_rules(), cr_parse_paths(), cr_path_spec(), cr_read_cellprofiler(), cr_read_cells(), cr_read_design(), cr_read_export(), cr_read_exports(), cr_read_qupath(), cr_read_segmantr(), cr_unit_map()

Examples

```
cells <- tibble::tibble(
  well_id = rep(c("u1", "u2"), each = 4),
  x = c(1, 2, 3, 4, 1, 2, 3, 90),
  y = c(1, 2, 3, 4, 1, 2, 3, 90)
)
# u1 and u2 share three of four centroids:
cr_centroid_overlap(cells, "u1", "u2")
```

---

cr_channels *List channels in a* cr_experiment

Description

List channels in a cr_experiment

Usage

```
cr_channels(experiment)
```

Arguments

experiment	A cr_experiment.

Value

A character vector of channel names.

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 10)
cr_channels(exp)
```

`cr_classify_result` *Deterministically classify numeric results using explicit boundaries*

Description

Deterministically classify numeric results using explicit boundaries

Usage

```
cr_classify_result(
  value,
  negative_upper,
  positive_lower,
  labels = c(negative = "NEGATIVE", indeterminate = "INDETERMINATE", positive =
    "POSITIVE")
)
```

Arguments

`value`	Numeric vector.
`negative_upper, positive_lower`	Explicit finite boundaries. Values at `negative_upper` are negative; values at `positive_lower` are positive; values between are indeterminate.
`labels`	Named character vector containing `negative`, `indeterminate`, and `positive`.

Value

Character vector with missing/non-finite inputs returned as `NA`.

`cr_column_map` *Declare a column contract for vendor exports*

Description

Segmented single-cell exports name their columns however the acquisition software happens to. A `cr_column_map` records how those raw headers translate to the analysis names used downstream, so that the mapping lives in one declarative object instead of being spread through reader code.

Usage

```
cr_column_map(exact = NULL, prefix = NULL, keep = NULL)

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

Arguments

`exact`	Named character vector. Names are raw header names, values the analysis name to rename them to.
`prefix`	Named character vector. Names are regular expressions matched against the raw header names, values the analysis name. The first matching column is renamed.
`keep`	Optional character vector of analysis names to retain after renaming, in this order. Names that are absent are ignored. `NULL` keeps every column.
`x`	A `cr_column_map`.
`...`	Ignored.

Details

Two matching modes are available because vendor headers are not always stable: `exact` matches a header verbatim, while `prefix` matches a regular expression. Prefix matching exists for headers that embed a unit glyph (an area column ending in a squared-micrometre symbol, for instance), where the exact spelling cannot be relied on.

Renaming is deliberately *tolerant*: names that do not occur in a given export are skipped silently rather than raising an error. Exports from the same instrument routinely carry different subsets of the available measurements, and a strict renamer forces a second reader to exist for every subset.

Value

An object of class `cr_column_map`: a list with elements `exact`, `prefix` and `keep`.

`x`, invisibly.

See Also

cr_read_export(), cr_read_exports().

Other import: cr_assign_units(), cr_centroid_overlap(), cr_extract_markers(), cr_filename_grammar(), cr_marker_rules(), cr_merge_rules(), cr_parse_paths(), cr_path_spec(), cr_read_cellprofiler(), cr_read_cells(), cr_read_design(), cr_read_export(), cr_read_exports(), cr_read_qupath(), cr_read_segmantr(), cr_unit_map()

Examples

```
map <- cr_column_map(
  exact = c(
    "Event Label" = "cell_id",
    "Signal - Mean Intensity" = "target_signal"
  ),
  prefix = c("^Nuclei - Area" = "area"),
  keep = c("cell_id", "target_signal", "area")
)
map
```

---

cr_compare_levels *Compare unit-level and cell-level effect estimates*

Description

Puts the two aggregation levels of a screen side by side and quantifies how much narrower the cell-level intervals are. A ratio well above one is pseudo-replication rather than precision: cells within a unit are not independent, so the cell-level interval understates the uncertainty of the very same estimate.

Usage

```
cr_compare_levels(unit_effects, cell_effects, by = NULL, estimate = "cohens_d")
```

Arguments

unit_effects	Effect grid computed with unit set, from cr_effect_grid().
cell_effects	Effect grid computed on the same data without unit.
by	Join keys. NULL (default) uses every non-numeric column the two grids share, which is the stratifying columns plus contrast. Keys must be unique in both grids. Include numeric strata explicitly when the default text keys do not uniquely identify a contrast.
estimate	Name of the effect size column to compare.

Value

A tibble with the join keys, estimate_unit, estimate_cell, width_unit, width_cell and ratio (width_unit / width_cell). The median ratio is attached as the median_ratio attribute.

See Also

cr_effect_grid().

Other effect sizes: cr_effect_grid(), cr_effect_size()

Examples

```
set.seed(1)
cells <- data.frame(
  compound = rep(c("CompoundA", "CompoundB"), each = 240),
  arm = rep(rep(c("reference", "interval_short"), each = 120), 2),
  unit_id = rep(paste0("u", 1:8), each = 60),
  log2_fc = stats::rnorm(480)
)
cells$log2_fc <- cells$log2_fc +
  ifelse(cells$compound == "CompoundA" & cells$arm == "interval_short",
         -1, 0)
u <- cr_effect_grid(cells, "log2_fc", "arm", "reference",
                    by = "compound", unit = "unit_id")
k <- cr_effect_grid(cells, "log2_fc", "arm", "reference",
                    by = "compound", min_n = 10)
cmp <- cr_compare_levels(u, k)
cmp
attr(cmp, "median_ratio")
```

---

cr_compute_metrics *Compute per-unit summary metrics*

Description

Returns a rich set of per-unit summary statistics for a single channel: mean, median, SD, MAD, CV, cell count and percent positive above a threshold. Useful as input for QC dashboards and as the between-unit variability input of a screen.

Usage

```
cr_compute_metrics(experiment, channel, positive_threshold = NULL, unit = NULL)
```

Arguments

experiment	A cr_experiment.
channel	Channel name (a column of experiment$cells).
positive_threshold	Optional numeric threshold. Cells above it are counted as "positive"; without it pct_positive is NA.
unit	Optional name of the column that identifies the analysis unit. May be a column of cells or of design. If NULL (default) the experiment's unit_var slot is used when present, and the spatial unit otherwise.

Value

A tibble with one row per analysis unit and the design columns appended.

See Also

`cr_summarize_wells()`, `cr_table_disposition()`.

Other quantification: `cr_fold_change()`, `cr_summarize_wells()`, `cr_table_disposition()`

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
cr_compute_metrics(exp, channel = "marker_1")
cr_compute_metrics(exp, channel = "marker_1", positive_threshold = 800)
```

---

cr_confusion_matrix *Confusion matrix for a logistic* cr_result

---

Description

Confusion matrix for a logistic `cr_result`

Usage

```
cr_confusion_matrix(result, threshold = 0.5)
```

Arguments

`result`	A `cr_result` from `cr_logistic()`.
`threshold`	Classification threshold on predicted probability (default `0.5`).

Value

A tibble with `sensitivity`, `specificity`, `ppv`, `npv`, `accuracy`, and the confusion-matrix counts.

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
res <- cr_logistic(exp, "marker_1", "CompoundA_high", "Untreated")
cr_confusion_matrix(res, 0.5)
```

cr_conservative_effect *Effect size at the confidence bound nearer the null*

Description

Returns the end of a confidence interval that lies closer to zero, which is the conservative effect a follow-up study should be powered on. When the interval spans the null there is nothing to power on and NA is returned.

Usage

```
cr_conservative_effect(ci_low, ci_high)
```

Arguments

ci_low	Numeric vector of lower interval bounds.
ci_high	Numeric vector of upper interval bounds.

Value

A numeric vector the length of the recycled inputs. Scalars can recycle; other lengths must agree, and lower bounds cannot exceed upper bounds.

See Also

cr_power(), cr_effect_grid().

Other power: cr_power(), cr_power_analysis(), cr_power_grid()

Examples

```
cr_conservative_effect(c(-1.8, -0.4), c(-0.6, 0.9))
```

cr_correct_batch *Correct batch effects*

Description

Applies a simple batch correction. "median_center" shifts each batch's median to the overall median. The legacy "combat" choice is refused: this single-channel interface cannot supply the feature-by-sample matrix required for a valid empirical-Bayes ComBat analysis. Previous versions could substitute median centering after a ComBat error. Select "median_center" explicitly if that is the intended analysis.

Usage

```
cr_correct_batch(
  experiment,
  batch_var,
  channel,
  method = c("median_center", "combat")
)
```

Arguments

experiment	A cr_experiment.
batch_var	Name of the batch variable, or a character vector of several column names that jointly define a batch (they are collapsed with cr_batch_key()). Columns are looked up in design first and then in cells.
channel	Channel to correct.
method	"median_center" or "combat".

Details

Batch correction removes a batch offset from the measured signal. It is *not* a substitute for cr_standardize_batch(), which expresses every cell relative to the control cells of its own batch and leaves the measured signal untouched.

Value

A modified cr_experiment.

See Also

cr_batch_key(), cr_standardize_batch()

Other batch standardization functions: cr_batch_key(), cr_batch_reference(), cr_standardize_batch()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
exp$design$batch <- rep(c("b1", "b2"), length.out = nrow(exp$design))
cr_correct_batch(exp, batch_var = "batch", channel = "marker_1")

# A batch defined by more than one column
exp$design$plate <- rep(c("P1", "P2"), length.out = nrow(exp$design))
cr_correct_batch(exp, batch_var = c("batch", "plate"),
                 channel = "marker_1")
```

`cr_dataset` *Build an ingested data set*

Description

A `cr_dataset` is what comes out of ingest: the cells of every export, the per-file provenance that lets any row be traced back to the acquisition it came from, and optionally the design those files encode. It is the object to inspect before committing to an analysis, and it converts to a `cr_experiment` with `cr_build_experiment()`.

Usage

```
cr_dataset(
  cells,
  design = NULL,
  unit_var = NULL,
  provenance = NULL,
  file_col = "source_path",
  metadata = list(),
  call = rlang::caller_env()
)

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

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

Arguments

`cells`	A data frame of cells, typically from `cr_read_exports()`.
`design`	Optional `cr_design()` object, or a data frame that is passed to `cr_design()`.
`unit_var`	Name of the analysis unit column. Defaults to the design's unit column, or the first of `well`, `slide`, `well_id` or `unit` present in `cells`.
`provenance`	Optional per-file table. When `NULL` it is derived from `cells`: one row per file with its cell count and every column that is constant within the file.
`file_col`	Name of the file column used for provenance. Default `"source_path"`.
`metadata`	Optional list of arbitrary user metadata.
`call`	The execution environment of the calling function. Used for error reporting; experts only.
`x`	A `cr_dataset`.
`...`	Ignored.
`object`	A `cr_dataset`.

Value

An object of class `cr_dataset`:

`cells` Tibble of per-cell measurements.

`provenance` Tibble with one row per source file, or `NULL`.

`design` A `cr_design`, or `NULL`.

`unit_var` Name of the analysis unit column, or `NULL`.

`metadata` List of user metadata.

`x`, invisibly.

A tibble with one row per source file, or `NULL` when the data set carries no provenance.

See Also

`cr_read_exports()`, `cr_design()`, `cr_build_experiment()`.

Other constructors: `cr_build_experiment()`, `cr_design()`, `cr_validate_experiment()`

Examples

```
cells <- tibble::tibble(
  source_path = rep(c("a.csv", "b.csv"), each = 3),
  source_file = rep(c("a.csv", "b.csv"), each = 3),
  well = rep(c("A01", "A02"), each = 3),
  treatment = rep(c("Vehicle", "CompoundA"), each = 3),
  target_signal = c(10, 12, 11, 30, 33, 29)
)
ds <- cr_dataset(cells)
ds
ds$provenance
```

---

cr_design *Build an experimental design object*

Description

A `cr_design` records both the design table — one row per analysis unit — and the roles its columns play: which column identifies the unit, which holds the treatment, which level of that treatment is the reference, and which columns together define a batch.

Usage

```
cr_design(
  data,
  unit = NULL,
  treatment = "treatment",
  control_level = NULL,
```

```
  batch_vars = NULL,
  levels = list(),
  keep = NULL,
  call = rlang::caller_env()
)

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

Arguments

data	A data frame carrying the design columns.
unit	Name of the column identifying the analysis unit. If NULL, the first of well, slide, well_id or unit present in data is used.
treatment	Name of the treatment (group) column. Default "treatment".
control_level	Optional value of treatment that is the reference level, for example the vehicle control.
batch_vars	Optional character vector of columns that together define a batch.
levels	Optional named list of factor level orders applied to the corresponding design columns.
keep	Optional character vector of design columns to keep. The unit column is always kept. NULL keeps every column of data.
call	The execution environment of the calling function. Used for error reporting; experts only.
x	A cr_design.
...	Ignored.

Details

Recording the roles alongside the table is what lets later stages standardise each cell against the control of *its own* batch rather than against one pooled reference. A batch is normally a combination of columns (compound, run, plate, experiment, pre-treatment interval), which a single batch variable cannot express.

data may be a per-unit table or a per-cell / per-file table; in the latter case it is collapsed to unique rows. If a unit maps to more than one combination of design values the collapse is ambiguous and the offending columns are named in the error, since silently keeping the first row would invent a design that was never run.

Value

An object of class cr_design:

table Tibble with one row per analysis unit.

unit Name of the unit column.

treatment Name of the treatment column.

control_level Reference level, or NULL.

`batch_vars` Character vector of batch columns.

`levels` The factor level orders that were applied.

x, invisibly.

See Also

`cr_dataset()`, `cr_build_experiment()`, `cr_read_design()`.

Other constructors: `cr_build_experiment()`, `cr_dataset()`, `cr_validate_experiment()`

Examples

```
units <- tibble::tibble(
  well = sprintf("A%02d", 1:6),
  treatment = rep(c("Vehicle", "CompoundA"), each = 3),
  plate = rep(c("Plate_1", "Plate_2"), 3),
  replicate = rep(1:3, 2)
)
design <- cr_design(units, control_level = "Vehicle",
                    batch_vars = c("plate"))
design
```

---

cr_dose_response *Fit a dose-response curve*

Description

Fits a dose-response model to well-level summaries of a treatment across doses. Supports 4-parameter log-logistic (`"4pl"`), 3-parameter log-logistic with fixed lower asymptote at zero (`"3pl"`), and a simple linear model (`"linear"`).

Usage

```
cr_dose_response(
  experiment,
  channel,
  treatment = NULL,
  model = c("4pl", "3pl", "linear"),
  log_dose = TRUE
)
```

Arguments

`experiment`	A `cr_experiment`.
`channel`	Channel name.
`treatment`	Character. Names of treatments to include (for example a compound at several doses). If `NULL`, the entire design is used after filtering by `dose > 0`.
`model`	`"4pl"`, `"3pl"` or `"linear"`.
`log_dose`	If `TRUE`, the fit is done on log10 doses. Doses must all be positive in that case.

Details

The 4PL model fitted is:

$$y = d + \frac{a - d}{1 + (x/e)^b}$$

where a is the top asymptote, d the bottom asymptote, e the inflection point (EC50 / IC50), and b the Hill slope.

Value

A list with class "cr_dose_response" containing the fitted model, the data used for the fit, the estimated parameters, and helper predictions.

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
# Add a synthetic dose-response sub-design
exp$design$dose <- ifelse(exp$design$treatment == "CompoundA_high",
                          500, exp$design$dose)
fit <- cr_dose_response(exp, channel = "marker_1", model = "4pl")
print(fit$params)
```

---

cr_effect_grid *Effect sizes for a whole grid of contrasts*

Description

Drives cr_effect_size() over every group in a screen: for each stratum given by by, each comparison level of group_var is contrasted against reference_level, and every requested effect size is reported with a confidence interval. The same function serves both aggregation levels: supply unit to aggregate the rows to one value per experimental unit (the unit of replication) before the contrast is computed, or leave it NULL to work on the rows as they are (typically single cells).

Usage

```
cr_effect_grid(
  data,
  value,
  group_var,
  reference_level,
  comparison_levels = NULL,
  by = NULL,
  unit = NULL,
  methods = c("cohens_d", "hedges_g", "cliffs_delta"),
  conf_level = 0.95,
  min_n = 3,
  test = c("t", "wilcox", "none"),
  p_adjust = c("bonferroni", "BH"),
  level = NULL
)
```

Arguments

data	A data frame (one row per cell or per unit) or a cr_experiment, in which case the design is joined onto the cells first.
value	Name of the numeric response column, for example a standardized log2 fold change or a raw channel.
group_var	Name of the column holding the compared groups.
reference_level	Level of group_var used as the reference arm of every contrast.
comparison_levels	Levels contrasted against the reference. NULL (default) uses every other level present, in factor-level order when group_var is a factor.
by	Optional character vector of stratifying columns. One set of contrasts is computed per combination, for example one per compound.
unit	Optional name of the column identifying the unit of replication. When given, rows are averaged to one value per unit before the contrast is computed.
methods	Effect sizes to compute; see cr_effect_size().
conf_level	Confidence level for the intervals.
min_n	Minimum number of observations required in *each* arm. Contrasts below it are skipped and recorded in the skipped attribute. Use 3 for unit level and something like 10 for cell level.
test	P-value to accompany the effect sizes: "t" (Welch two-sample t-test, the default), "wilcox" or "none".
p_adjust	Multiplicity adjustments applied across the grid. Default both "bonferroni" and "BH"; see stats::p.adjust().
level	Optional label written into the level column. Defaults to "unit" when unit is given and "cell" otherwise.

Details

Cell-level intervals are anticonservative by construction, because cells within a unit are not independent. Compute both levels and pass them to cr_compare_levels() to quantify the difference; interpretation belongs to the unit level.

Multiplicity adjustment is applied across the entire returned grid and reported *alongside* the unadjusted p-value, never in place of it: for an exploratory screen the effect sizes and their intervals are the reportable quantity.

Value

A tibble with one row per stratum and contrast: the by columns, contrast, level, n_ref, n_cmp, mean_shift, one <method>, <method>_ci_low, <method>_ci_high and <method>_magnitude triplet per requested method, p_value, one p_<adjustment> column per entry of p_adjust, and ci_excludes_zero for the first method. The attributes conf_level, level, primary_method and skipped (a tibble of the contrasts that fell below min_n) are attached.

See Also

cr_effect_size(), cr_compare_levels(), cr_power_grid(), cr_blocked_effect().

Other effect sizes: cr_compare_levels(), cr_effect_size()

Examples

```
set.seed(1)
units <- data.frame(
  compound = rep(c("CompoundA", "CompoundB"), each = 12),
  arm = rep(rep(c("reference", "interval_short"), each = 6), 2),
  unit_id = paste0("u", 1:24),
  log2_fc = c(stats::rnorm(6, 0, 0.3), stats::rnorm(6, -1, 0.3),
              stats::rnorm(6, 0, 0.3), stats::rnorm(6, -0.1, 0.3))
)
eff <- cr_effect_grid(units, value = "log2_fc", group_var = "arm",
                      reference_level = "reference", by = "compound")
eff[, c("compound", "contrast", "cohens_d", "cohens_d_ci_low",
        "cohens_d_ci_high", "ci_excludes_zero")]
```

---

cr_effect_size *Compute effect sizes between two samples*

Description

Standardized effect sizes for a comparison of two independent samples, each with a confidence interval. Confidence intervals are analytic by default (deterministic and cheap on large samples); a bootstrap interval is available for cases where the analytic approximation is not wanted.

Usage

```
cr_effect_size(
  x,
  y,
 method = c("cohens_d", "hedges_g", "cliffs_delta", "rank_biserial", "glass_delta"),
  ci = 0.95,
  ci_method = c("analytic", "bootstrap"),
  n_boot = 200,
  seed = NULL
)
```

Arguments

x	Numeric vector (treatment group).
y	Numeric vector (control / reference group).
method	Character vector of methods to compute. Allowed: "cohens_d", "hedges_g", "cliffs_delta", "rank_biserial", "glass_delta".

ci	Confidence level for the interval (default 0.95). Set NULL to skip interval estimation.
ci_method	"analytic" (default) or "bootstrap".
n_boot	Number of bootstrap resamples used when ci_method = "bootstrap".
seed	Optional integer seed for the bootstrap. The global random number stream is restored on exit.

Details

Cohen's *d* uses the pooled standard deviation without small-sample correction and its interval is the large-sample (Hedges-Olkin) approximation with a *t* quantile on n_x + n_y - 2 degrees of freedom. Hedges' g applies the bias correction $J = 1 - 3/(4 \cdot df - 1)$ to both estimate and interval. Glass's delta standardizes by the standard deviation of y only. Cliff's delta uses the consistent variance estimator with the interval on the transformed scale, so the bounds always lie inside $[-1, 1]$. The rank-biserial correlation of two independent samples equals Cliff's delta and is reported on the same scale.

Value

A tibble with columns method, estimate, ci_low, ci_high and magnitude.

See Also

cr_effect_grid() for a whole grid of contrasts, cr_power() for the sample size implied by an effect size.

Other effect sizes: cr_compare_levels(), cr_effect_grid()

Examples

```
set.seed(1)
cr_effect_size(stats::rnorm(100, 1), stats::rnorm(100, 0))

# bootstrap interval with a reproducible seed
cr_effect_size(stats::rnorm(50, 1), stats::rnorm(50, 0),
               method = "cliffs_delta",
               ci_method = "bootstrap", n_boot = 50, seed = 42)
```

---

cr_enumerate *Write a vector out as an English list*

Description

Renders a character vector as "a", "a and b" or "a, b and c", with a fixed placeholder for the empty case. Sentences that enumerate results — which contrasts cleared a threshold, which arms were excluded — should be generated for the same reason the numbers are: the hand-written version is right until the analysis changes.

Usage

```
cr_enumerate(x, conjunction = "and", empty = "none", oxford = FALSE)
```

Arguments

x	Character vector. NA and empty strings are dropped.
conjunction	Word joining the last two elements.
empty	Text returned when nothing is left.
oxford	Whether to place a comma before the conjunction.

Value

A single string.

See Also

cr_macros(), cr_format_number().

Other macros: cr_format_number(), cr_macro_name(), cr_macros(), cr_macros_from()

Examples

```
cr_enumerate(c("CompoundA", "CompoundB", "CompoundC"))
cr_enumerate(c("CompoundA", "CompoundB"), conjunction = "or")
cr_enumerate(character())
```

---

cr_example_design *Generate an example experimental design*

Description

Builds the unit-level design table used by cr_example_experiment(). Six treatment levels are crossed with two plates and two pre-treatment intervals, so that a batch is the *combination* of plate and interval rather than a single column and every batch still contains untreated units to standardise against.

Usage

```
cr_example_design(plate_format = 96, n_wells_per_replicate = 4)
```

Arguments

plate_format	Either 96 or 384.
n_wells_per_replicate	Number of units per replicate per treatment level.

Details

Treatment names are deliberately abstract. `Untreated` is the vehicle level, `PosControl` a positive control, and `CompoundA` appears at a low and a high exposure level so that a dose axis exists; `CompoundB` and `CompoundC` are further compounds at the low exposure level. The four distinct values of `dose` are the four exposure levels of the demonstration assay.

Value

A tibble with one row per unit and the columns `well`, `treatment`, `dose`, `dose_unit`, `group`, `replicate`, `plate`, `interval` and `timepoint`.

See Also

`cr_example_experiment()`, `cr_example_screen()`.

Other example data: `cr_example_experiment()`, `cr_example_exports()`, `cr_example_files()`, `cr_example_path()`, `cr_example_screen()`

Examples

```
design <- cr_example_design(96)
table(design$treatment, design$interval)
```

---

cr_example_experiment *Generate a synthetic* cr_experiment

Description

Simulates a compact demonstration experiment: one 96-unit plate format, six treatment levels, four exposure levels, two plates and two pre-treatment intervals. It is small enough that every example in the package can build a fresh copy in well under a second, and structured enough that the batch, quality-control and effect-size functions all have something to work on.

Usage

```
cr_example_experiment(
  seed = 42,
  n_cells_per_well = 150,
  n_wells_per_replicate = 4
)
```

Arguments

`seed`	Random seed. `NULL` uses the current RNG state.
`n_cells_per_well`	Mean number of cells per unit (Poisson, with a floor of ten).
`n_wells_per_replicate`	Number of units per replicate.

Details

Use `cr_example_screen()` instead when a multi-compound screen with per-batch controls, merged analysis units and a specificity arm is needed.

Value

A `cr_experiment` whose `cells` table holds one row per cell (`cell_id`, `well`, `x`, `y`, `area`, `circularity`, `DAPI`, `marker_1`, `marker_2`, `marker_3`) and whose `design` table holds one row per unit, as described in `cr_example_design()`.

Generating model

Values are drawn independently per cell from log-normal distributions, so that the marker channels are right-skewed the way measured intensities are:

unit intercept Each unit draws $u \sim N(0, 0.15^2)$, added to the log mean of every channel in that unit. This is what makes the unit, and not the cell, the honest replicate.

target signal `marker_1` is $\mathrm{LogNormal}(\log 500 + u + e \log 2,\ 0.5^2)$, where e is the treatment effect in log2 units: `0` untreated, `1.0` and `3.0` for the low and high exposure level of `CompoundA`, `3.3` for the positive control, `0.4` for `CompoundB` and `0.7` for `CompoundC`.

further channels `marker_2` moves against the target signal at $-0.4e$, `marker_3` weakly with it at $0.3e$.

nuclear stain `DAPI` is $\mathrm{LogNormal}(\log 500 + u,\ 0.3^2)$ and carries no treatment effect.

morphology Nuclear `area` is $\mathrm{LogNormal}(\log 400,\ 0.25^2)$ and `circularity` is $\mathrm{Beta}(5, 2)$.

plate position Units on the outer rows and columns get a fixed `+0.07` log bump on every channel, an edge effect for the plate map to show.

debris Five per cent of the cells of each unit are shrunk to a twentieth of their area and a tenth of their nuclear signal, so that the quality-control filters have something to remove.

artefacts Two named units are multiplied up on `marker_1` and down on `area`, a saturated pair for the outlier screens.

The random number generator state of the caller is restored on exit, so calling this function does not disturb a seeded analysis.

See Also

`cr_example_design()`, `cr_example_screen()`, `cr_example_files()`.

Other example data: `cr_example_design()`, `cr_example_exports()`, `cr_example_files()`, `cr_example_path()`, `cr_example_screen()`

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 20)
exp
head(exp$design, 3)
```

`cr_example_exports` *Write a synthetic export tree*

Description

Writes a nested directory of single-acquisition export files whose design information lives in the directory layout and the file names rather than inside the files. This is the input the multi-file ingest functions were written for, so that `cr_read_exports()`, `cr_parse_paths()` and `cr_extract_markers()` can be demonstrated on real files.

Usage

```
cr_example_exports(
  dir = tempdir(),
  seed = 42,
  n_cells = 20,
  format = c("csv", "xlsx")
)
```

Arguments

`dir`	Directory to write the tree into. Created when it does not exist.
`seed`	Random seed. `NULL` uses the current RNG state.
`n_cells`	Number of cells written per file.
`format`	File format, `"csv"` (default) or `"xlsx"`. Writing `"xlsx"` needs the `writexl` package.

Details

The layout below `dir` is `Run1/<compound>/<experiment>/<plate>/<file>`, and the file names follow the grammar `<compound>_<interval>_<dose>_<mode>_<replicate>` in which an absent token is meaningful: a name carrying no exposure token is a vehicle control. Parenthetical markers are attached to three files – a two-pass acquisition, a repeated read and a reagent-omitted acquisition – and one directory is marked as a partly filled plate.

Column headers are raw instrument names rather than analysis names, and one of them carries a unit glyph, so that a `cr_column_map()` is genuinely required to read them.

Value

A character vector of the written file paths, invisibly.

See Also

`cr_read_exports()`, `cr_path_spec()`, `cr_example_screen()`, `cr_example_path()`.

Other example data: `cr_example_design()`, `cr_example_experiment()`, `cr_example_files()`, `cr_example_path()`, `cr_example_screen()`

Examples

```
d <- file.path(tempdir(), "cr_example_exports")
files <- cr_example_exports(d, seed = 1, n_cells = 5)
basename(files)

map <- cr_column_map(
  exact = c("Event Label" = "cell_id",
            "Target - Signal Mean" = "target_signal"),
  prefix = c("^Nuclei - Area" = "area")
)
cells <- cr_read_exports(d, column_map = map, progress = FALSE)
head(cells[, c("source_file", "cell_id", "target_signal", "area")], 3)
```

`cr_example_files` *Write example files in several on-disk formats*

Description

Writes the tables behind `cr_example_experiment()` in the layouts the `cr_read_*` importers understand, so that the import functions can be demonstrated on real files without any being shipped.

Usage

```
cr_example_files(dir = tempdir(), seed = 42)
```

Arguments

`dir`	Directory to write into. Created when it does not exist.
`seed`	Seed passed to `cr_example_experiment()`.

Value

A character vector of the written file paths, invisibly.

See Also

`cr_example_exports()` for a nested export tree with the design encoded in the paths, `cr_example_path()` for the files shipped inside the package.

Other example data: `cr_example_design()`, `cr_example_experiment()`, `cr_example_exports()`, `cr_example_path()`, `cr_example_screen()`

Examples

```
d <- file.path(tempdir(), "cr_example_files")
files <- cr_example_files(d, seed = 1)
basename(files)
```

`cr_example_path` *Locate the example files shipped with the package*

Description

Three tiny synthetic files are installed under `extdata` so that examples, tests and vignettes can read a real file without writing one first and without depending on any gated data.

Usage

```
cr_example_path(file = NULL)
```

Arguments

`file`	Name of one shipped file, or `NULL` (the default) to return them all.

Details

`example-export.csv` One acquisition of one unit as a vendor export: raw instrument headers, an all-blank trailing row of the kind many instruments write, and a header carrying a unit glyph, which is what the prefix matching of `cr_column_map()` exists for.

`example-export.xlsx` The same acquisition as a workbook.

`example-design.csv` The unit-level design table covering the units in the exports.

The two exports are the treated acquisition that `cr_example_exports(seed = 42, n_cells = 20)` writes, so the shipped files can be regenerated at any time rather than being opaque binaries.

Value

A character vector of absolute paths; length one when `file` is given.

See Also

`cr_read_export()`, `cr_read_design()`, `cr_example_files()`.

Other example data: `cr_example_design()`, `cr_example_experiment()`, `cr_example_exports()`, `cr_example_files()`, `cr_example_screen()`

Examples

```
basename(cr_example_path())

cells <- cr_read_export(cr_example_path("example-export.csv"))
head(cells, 3)
```

`cr_example_screen` *Generate a synthetic multi-compound screen*

Description

Simulates a screen of up to ten compounds, each acquired at four exposure levels and two pre-treatment intervals, on two plates and in two experiments. Unlike `cr_example_experiment()` the analysis unit is not the raw plate well: units carry file provenance, some are assembled from more than one acquisition, and one arm is set aside instead of screened.

Usage

```
cr_example_screen(
  seed = 42,
  n_compounds = 10,
  n_cells_per_well = 40,
  n_units_per_arm = 3,
  n_experiments = 2
)
```

Arguments

`seed`	Random seed. `NULL` uses the current RNG state.
`n_compounds`	Number of compounds, 1 to 10. They are named `CompoundA` to `CompoundJ`.
`n_cells_per_well`	Mean number of cells per unit (Poisson, with a floor of ten).
`n_units_per_arm`	Units per compound, experiment, interval and exposure level. Three or more is needed for the effect-size grid.
`n_experiments`	Number of experiments, 1 or 2.

Details

Everything the screening functions need is present: a vehicle control inside every batch for `cr_batch_reference()` and `cr_standardize_batch()`, one unit that fails a control-referenced gate for `cr_qc_gate()`, enough units per arm for `cr_effect_grid()`, plates crossed with arms for `cr_blocked_effect()`, and a reagent-omitted arm in `set_aside`.

Value

A `cr_experiment` with

`cells` one row per cell: `cell_id`, `well_id`, `source_file`, `source_path`, `x`, `y`, `area`, `circularity`, `nuclear_signal`, `target_signal`.

`design` one row per unit: `well_id`, `compound`, `experiment`, `plate`, `well`, `interval`, `dose`, `dose_unit`, `treatment`, `group`, `replicate`.

`provenance` one row per acquisition file, with marker flags and the number of files each unit was assembled from.

`set_aside` a list whose `reagent_omitted` element holds the specificity arm.

`unit_var` is `"well_id"` and `batch_vars` is `c("compound", "experiment", "plate", "interval")`.

Generating model

A batch is the combination of `compound`, `experiment`, `plate` and `interval`. Writing p_c for the potency of compound c, d for the exposure level and u_w for the unit intercept:

potency p_c runs from `1.6` down to `0.1` in equal steps from `CompoundA` to the last compound, so the rank order of the screen is known in advance and the top hit is a hit by construction rather than by chance.

exposure The response saturates in dose as $d/(d+40)$, so the four exposure levels are not equally spaced in effect.

interval The longer pre-treatment interval multiplies the effect by `1.15`.

effect $e = p_c \cdot d/(d+40) \cdot g_{interval}$, in log2 units against the vehicle of the same batch.

unit intercept $u_w \sim N(0, 0.12^2)$ on the log scale, one draw per unit.

acquisition offsets The second experiment has a `1.35` times higher raw baseline and the second plate a `1.08` times higher one. Both act on control and treated units alike, which is what standardising against the control of a unit's own batch removes, and why raw signal is not comparable across experiments.

target signal $\mathrm{LogNormal}(\log(400 \cdot g_{exp} \cdot g_{plate}) + u_w + e \log 2,\ 0.45^2)$.

nuclear signal $\mathrm{LogNormal}(\log 600 + u_w,\ 0.3^2)$, with no treatment effect.

morphology Nuclear `area` is $\mathrm{LogNormal}(\log 180,\ 0.3^2)$ and `circularity` is $\mathrm{Beta}(6, 2)$. Four per cent of objects are shrunk to an eighth of their area, the sub-threshold debris that `cr_exclude_small()` is meant to drop.

Planted edge cases

gate failure One treated unit of the first compound is given a negative effect, so it sits below the control of its own batch and fails a control-referenced gate.

two-pass acquisition One unit's cells are split across two files, the second carrying a `(split)` marker and a `.1` replicate suffix. Left unmerged it would count twice.

repeated read One unit's cells are split across a file and its `(repeat)` sibling, which has to merge into it.

look-alike suffix One separate unit is named with a `.2` replicate suffix, which must *not* merge with the unit named `2`: it is a different physical unit on a different plate.

arm outside the screen A reagent-omitted arm is generated and placed in `set_aside` rather than in the analysis pool.

The random number generator state of the caller is restored on exit.

See Also

cr_example_experiment(), cr_example_exports(), cr_batch_reference(), cr_effect_grid().

Other example data: cr_example_design(), cr_example_experiment(), cr_example_exports(), cr_example_files(), cr_example_path()

Examples

```
screen <- cr_example_screen(seed = 1, n_compounds = 3,
                            n_cells_per_well = 12)
screen
table(screen$design$compound, screen$design$treatment)

# Units assembled from more than one acquisition file
prov <- screen$provenance
unique(prov$well_id[prov$n_files > 1])
```

---

cr_exclude_small *Exclude sub-threshold objects with a data-derived cut-off*

Description

Removes objects whose segmentation size falls below a quantile of the observed size distribution. Unlike cr_qc_filter(), which applies an absolute threshold, the cut-off here is derived from the data, so it adapts to magnification and segmentation settings. The realised threshold is recorded, because a data-derived cut-off is only reproducible if the value it resolved to is reported.

Usage

```
cr_exclude_small(
  experiment,
  var = "area",
  probs = 0.1,
  threshold = NULL,
  scope = c("pooled", "batch"),
  batch_vars = NULL
)
```

Arguments

experiment	A cr_experiment.
var	Name of the size column in cells. Default "area".
probs	Quantile of the var distribution used as the threshold (0-1). Default 0.10, i.e. the lowest tenth is dropped. Ignored when threshold is supplied.
threshold	Optional absolute threshold. When supplied it overrides probs and is used for every batch.

scope	"pooled" (default) computes a single threshold from all cells; "batch" computes one threshold per batch.
batch_vars	Character vector of columns (from cells or design) that define a batch. Required when scope = "batch".

Details

Cells with a non-finite value in var are always removed: they cannot be placed on either side of the threshold.

scope = "pooled" computes one threshold from all cells. That is the right choice when the segmentation settings were shared across the whole study. scope = "batch" computes one threshold per combination of batch_vars, which protects a batch acquired at a different magnification from being trimmed against a pool it does not belong to — at the cost of a threshold that is no longer comparable between batches.

Run this step *before* cr_balance_cells(): balancing changes how many cells each unit contributes to the pool, so a threshold computed afterwards is taken on a differently weighted distribution.

Value

A modified cr_experiment. The realised threshold(s) are stored in metadata$exclude_small as a tibble and summarised in the QC log.

See Also

cr_balance_cells(), cr_qc_filter(), cr_qc_report().

Other quality control: cr_apply_gate(), cr_balance_cells(), cr_qc_doublets(), cr_qc_filter(), cr_qc_gate(), cr_qc_gate_impact(), cr_qc_intensity(), cr_qc_manual(), cr_qc_report(), cr_qc_summary(), print.cr_qc_gate()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 40)
exp2 <- cr_exclude_small(exp, var = "area", probs = 0.10)
exp2$metadata$exclude_small

# one threshold per replicate block
exp3 <- cr_exclude_small(exp, scope = "batch", batch_vars = "replicate")
exp3$metadata$exclude_small
```

---

cr_export_plots *Export plots to PNG, PDF or SVG in batch*

Description

Export plots to PNG, PDF or SVG in batch

Usage

```
cr_export_plots(
  plots,
  path,
  format = c("png", "pdf", "svg"),
  width = 6,
  height = 4,
  dpi = 300,
  ...
)
```

Arguments

plots	A named list of ggplot2 objects. Unnamed elements are numbered.
path	Output directory, created when needed.
format	"png", "pdf" or "svg".
width, height	Figure dimensions in inches.
dpi	Resolution for raster outputs.
...	Further arguments passed to ggplot2::ggsave().

Value

A character vector of written file paths (invisibly).

See Also

cr_export_tables().

Other reporting: cr_export_results(), cr_export_tables(), cr_render_report(), cr_report(), cr_table_qc(), cr_tables()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 20)
p1 <- cr_plot_intensity(exp, "marker_1")
out <- tempfile("plots_")
cr_export_plots(list(intensity = p1), out, width = 4, height = 3)
```

---

cr_export_report_audit

Export report audit provenance as JSON

Description

Export report audit provenance as JSON

Usage

```
cr_export_report_audit(report, path, output_file = NULL)
```

Arguments

report	A cr_lab_report.
path	Output JSON path.
output_file	Optional rendered file to hash.

---

cr_export_report_spec *Export a report specification as versioned JSON*

Description

Named field labels are encoded as a JSON object. Date-time export retains its existing whole-second representation; fractional precision is not preserved by this format.

Usage

```
cr_export_report_spec(spec, path, pretty = TRUE)
```

Arguments

spec	A cr_report_spec.
path	Output JSON path.
pretty	Pretty-print JSON.

---

cr_export_results *Export results to CSV, Excel or RDS*

Description

Writes a single flat table of results. Anything the reporting layer produces is accepted: a cr_report, a cr_result, the list returned by cr_test_all(), a named list of tables, or a plain data frame.

Usage

```
cr_export_results(results, path, format = NULL)
```

Arguments

results	A cr_report, a cr_result, a list of cr_result objects, a named list of data frames, or a data frame.
path	Output file path. The extension determines the format unless format is given.
format	One of "csv", "xlsx", "rds". NULL (default) infers it from the extension of path, falling back to "csv".

Value

The output path (invisibly).

See Also

`cr_export_tables()` to write several tables at once and `cr_macros()` to emit individual numbers.

Other reporting: `cr_export_plots()`, `cr_export_tables()`, `cr_render_report()`, `cr_report()`, `cr_table_qc()`, `cr_tables()`

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 20)
rep <- cr_report(exp)
f <- tempfile(fileext = ".csv")
cr_export_results(rep$tables$disposition, f)
utils::read.csv(f)
```

---

cr_export_tables *Export a set of tables to CSV or Excel*

Description

Writes the named list produced by `cr_tables()`. Excel output can go into one workbook with a sheet per table; CSV always writes one file per table into a directory, because a CSV file holds exactly one table.

Usage

```
cr_export_tables(tables, path, format = c("csv", "xlsx"), one_file = TRUE)
```

Arguments

`tables`	A named list of data frames, or a single data frame.
`path`	Destination. For a single Excel workbook this is the file path; otherwise it is a directory, which is created when needed.
`format`	`"csv"` (default) or `"xlsx"`.
`one_file`	Whether to write one Excel workbook with a sheet per table. Ignored for CSV.

Value

A character vector of written paths (invisibly).

See Also

`cr_tables()`, `cr_export_results()`.

Other reporting: `cr_export_plots()`, `cr_export_results()`, `cr_render_report()`, `cr_report()`, `cr_table_qc()`, `cr_tables()`

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 20)
tabs <- cr_tables(exp)
out <- tempfile("tables_")
paths <- cr_export_tables(tabs, out)
basename(paths)
```

`cr_extract_markers` *Extract file-name markers into typed flags*

Description

Applies a `cr_marker_rules()` set to a table that carries a file-name column, adding one logical column per rule plus a character `variant` column holding any unmatched trailing parenthetical.

Usage

```
cr_extract_markers(
  x,
  name_col = "source_file",
  container_col = NULL,
  rules = cr_marker_rules(),
  stem_col = NULL,
  strip_ext = TRUE,
  call = rlang::caller_env()
)
```

Arguments

`x`	A data frame with one row per file (or per cell).
`name_col`	Name of the column holding the file name. Default `"source_file"`.
`container_col`	Optional name of the column holding the container (plate or directory) name that `partial_plate` is matched against.
`rules`	A `cr_marker_rules()` object.
`stem_col`	Optional name of a column to write the marker-stripped name into.
`strip_ext`	Logical. Remove a trailing file extension before matching. Default `TRUE`.
`call`	The execution environment of the calling function. Used for error reporting; experts only.

Value

`x` with the columns `merge_unit`, `partial_plate`, `omitted_reagent`, `reacquisition`, `lot` (all logical) and `variant` (character) added. Columns for rules that were not supplied are `FALSE` throughout.

See Also

cr_marker_rules(), cr_parse_paths().

Other import: cr_assign_units(), cr_centroid_overlap(), cr_column_map(), cr_filename_grammar(), cr_marker_rules(), cr_merge_rules(), cr_parse_paths(), cr_path_spec(), cr_read_cellprofiler(), cr_read_cells(), cr_read_design(), cr_read_export(), cr_read_exports(), cr_read_qupath(), cr_read_segmantr(), cr_unit_map()

Examples

```
files <- tibble::tibble(
  source_file = c("CompoundA_10uM_treated_1.csv",
                  "CompoundA_10uM_treated_1.1 (split).csv",
                  "CompoundA_vehicle_1 (no reagent).csv",
                  "CompoundA_vehicle_2 (weekend).csv"),
  plate = c("Plate_1", "Plate_1", "Plate_1", "Plate_2 (partial)")
)
cr_extract_markers(
  files,
  container_col = "plate",
  rules = cr_marker_rules(merge_unit = "\\(split\\)",
                          partial_plate = "\\(partial\\)",
                          omitted_reagent = "\\(no reagent\\)")
)
```

---

cr_figure_caption *Build a descriptive figure caption*

Description

Build a descriptive figure caption

Usage

```
cr_figure_caption(
  title = NULL,
  description = NULL,
  n = NULL,
  unit = NULL,
  qc_note = NULL
)
```

Arguments

title, description, qc_note	User-supplied descriptive text.
n	Optional observation count.
unit	Optional unit-of-replication label.

cr_figure_size *Standard deterministic figure dimensions*

Description

Standard deterministic figure dimensions

Usage

```
cr_figure_size(
  size = c("single", "double", "report", "square"),
  units = c("mm", "in")
)
```

Arguments

size	One of single, double, report, or square.
units	"mm" or "in".

cr_filename_grammar *Declare the token grammar of an export file name*

Description

A cr_filename_grammar states how one export file name decomposes into design facts: which trailing token is the replicate index, which leading labels are stripped, which substrings are tokens, and which whole-name shapes are legal at all.

Usage

```
cr_filename_grammar(
  tokens = list(),
  defaults = list(),
  core_patterns = character(),
  typo_fixes = character(),
  prefix_strip = character(),
  replicate = "[0-9]+(?:\\.[0-9]+)?",
  sep = "_",
  normalise_space = TRUE
)

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

Arguments

tokens	Named list (or named character vector) of regular expressions. Each is searched for in the name core; the matched text becomes the value of a column named after the list element.
defaults	Named list of values to use when a token is absent. Elements with no entry default to NA.
core_patterns	Character vector of regular expressions. The core (the name with extension, markers, replicate index and prefix removed) must match at least one of them. Empty means no whitelist.
typo_fixes	Named character vector of pattern = replacement repairs applied before anything else is parsed.
prefix_strip	Character vector of leading labels to strip from the core, longest first.
replicate	Regular expression for the trailing replicate token. Default "[0-9]+(?:\\.[0-9]+)?", which accepts both 1 and 1.1.
sep	Token separator. Default "_".
normalise_space	Logical. Convert runs of whitespace to sep and collapse repeated separators. Default TRUE.
x	A cr_filename_grammar.
...	Ignored.

Details

The core_patterns whitelist exists because the absence of a token is itself meaningful — a name with no interval token is the untreated reference arm, a name with no concentration token is a vehicle control. A mistyped token therefore matches nothing, falls through to the defaults, and silently reclassifies a treated unit as a vehicle control, pulling it into the control denominator of its own batch. With a whitelist in place such a name is a hard error instead.

prefix_strip entries are removed longest-first, so a short label cannot consume a longer one that starts with the same characters.

Value

An object of class cr_filename_grammar (a list).

x, invisibly.

See Also

cr_parse_paths(), cr_path_spec().

Other import: cr_assign_units(), cr_centroid_overlap(), cr_column_map(), cr_extract_markers(), cr_marker_rules(), cr_merge_rules(), cr_parse_paths(), cr_path_spec(), cr_read_cellprofiler(), cr_read_cells(), cr_read_design(), cr_read_export(), cr_read_exports(), cr_read_qupath(), cr_read_segmantr(), cr_unit_map()

Examples

```
g <- cr_filename_grammar(
  tokens = list(interval = "[0-9]+min", dose = "[0-9]+uM",
                mode = "treated|vehicle"),
  defaults = list(interval = "none", dose = "vehicle"),
  core_patterns = c("^vehicle$",
                    "^[0-9]+min_vehicle$",
                    "^[0-9]+uM_treated$",
                    "^[0-9]+min_[0-9]+uM_treated$"),
  prefix_strip = c("CompoundA", "CompoundB")
)
g
```

---

cr_filter_cells *Filter cells in a* cr_experiment

Description

A thin wrapper around `dplyr::filter()` applied to the `cells` slot. Filtering is additive: rows excluded here are removed from the returned experiment and logged in `qc_log`.

Usage

```
cr_filter_cells(experiment, ...)
```

Arguments

`experiment`	A `cr_experiment`.
`...`	Filtering expressions passed on to `dplyr::filter()`.

Value

A modified `cr_experiment`.

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 10)
cr_filter_cells(exp, area > 50)
```

cr_finalize_result *Apply a caller-selected QC finalization rule*

Description

Apply a caller-selected QC finalization rule

Usage

```
cr_finalize_result(
  classification,
  qc_status,
  failed_statuses = "FAIL",
  invalid_label = "INVALID",
  invalidate = TRUE
)
```

Arguments

`classification`	Character vector.
`qc_status`	Character vector, recycled against `classification`.
`failed_statuses`	QC labels treated as failed.
`invalid_label`	Label used for failed QC.
`invalidate`	Whether failed QC replaces the supplied classification.

cr_fold_change *Compute fold change relative to a control group*

Description

Returns log2 fold changes at the cell and unit levels, relative to one pooled reference computed from the control group.

Usage

```
cr_fold_change(
  experiment,
  channel,
  control_group,
  method = c("median", "mean"),
  eps = 0,
  unit = NULL
)
```

Arguments

experiment	A cr_experiment.
channel	Channel name (a column of experiment$cells).
control_group	Value in design$treatment that defines the reference.
method	"median" (default) or "mean" — the estimator used to aggregate within a unit before taking log2 ratios.
eps	Additive offset applied to numerator and denominator before the ratio is taken, i.e. log2((value + eps) / (ref + eps)). The default 0 keeps the historical behavior, which floors the numerator at a small positive value instead. An additive offset is the better choice for counts and for signals that legitimately reach zero, because flooring turns every zero into the same arbitrary large negative fold change.
unit	Optional name of the column that identifies the analysis unit. May be a column of cells or of design. If NULL (default) the experiment's unit_var slot is used when present, and the spatial unit otherwise.

Details

The reference is pooled across the whole experiment. When controls differ systematically between plates, runs or acquisition days, standardize each cell against the control of its own batch instead — a pooled reference silently mixes those batches together.

Value

A list with components cell (tibble, one row per cell with log2_fc), well (one row per analysis unit) and summary (per treatment: number of units, median, mean and SD of the unit-level log2 fold change).

See Also

cr_summarize_wells().

Other quantification: cr_compute_metrics(), cr_summarize_wells(), cr_table_disposition()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
fc <- cr_fold_change(exp, channel = "marker_1",
                     control_group = "Untreated")
fc$summary

# additive offset instead of a floor
fc2 <- cr_fold_change(exp, channel = "marker_1",
                      control_group = "Untreated", eps = 1)
fc2$summary
```

cr_format_interval *Format an estimate and confidence interval*

Description

Format an estimate and confidence interval

Usage

```
cr_format_interval(estimate, lower, upper, digits = 2)
```

Arguments

estimate, lower, upper	Numeric vectors.
digits	Decimal places.

cr_format_number *Format a number for a generated document*

Description

Fixed-decimal formatting with the conventions a manuscript needs: a placeholder rather than NaN for non-finite values, an optional explicit sign — never printed on an exact zero, where a sign would claim a direction the data does not support — and thousands separators for counts.

Usage

```
cr_format_number(x, digits = 3, signed = FALSE, big_mark = ",", na = "--")
```

Arguments

x	Numeric vector.
digits	Decimal places.
signed	Whether to print a leading + on positive values.
big_mark	Thousands separator.
na	Placeholder for NA, NaN and infinite values.

Value

A character vector of the same length as x.

See Also

cr_macros(), cr_enumerate().

Other macros: cr_enumerate(), cr_macro_name(), cr_macros(), cr_macros_from()

Examples

```
cr_format_number(c(1.2345, -0.5, 0, NA))
cr_format_number(c(1.2345, -0.5, 0), signed = TRUE)
cr_format_number(128400, digits = 0)
```

---

cr_format_percent *Format proportions as percentages*

Description

Format proportions as percentages

Usage

```
cr_format_percent(x, digits = 1)
```

Arguments

x	Numeric proportions.
digits	Decimal places.

---

cr_format_pvalue *Format p-values without displaying calculated zero*

Description

Format p-values without displaying calculated zero

Usage

```
cr_format_pvalue(x, digits = 3)
```

Arguments

x	Numeric p-values.
digits	Decimal places determining the reporting threshold.

cr_ic50 *Extract IC50 / EC50 from a dose-response fit*

Description

Extract IC50 / EC50 from a dose-response fit

Usage

```
cr_ic50(fit, level = 0.95)
```

Arguments

fit	A cr_dose_response returned by cr_dose_response().
level	Confidence level for the interval (default 0.95).

Value

A tibble with estimate, ci_low, ci_high, parameter (IC50 or EC50) and units.

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
exp$design$dose <- ifelse(exp$design$treatment == "CompoundA_high",
                          500, exp$design$dose)
fit <- cr_dose_response(exp, channel = "marker_1", model = "4pl")
cr_ic50(fit)
```

cr_import_report_spec *Import a versioned report specification from JSON*

Description

Date-only timestamps retain Date class; ISO timestamps with Z or numeric offsets are restored as UTC date-times. Legacy space-separated UTC times with minutes or seconds (optionally ending in UTC) are also accepted. Custom-section lists and named field-label objects retain their structure. This cannot recover names or fractional seconds lost in older exports and does not guarantee equality of whole-report hashes after a JSON round trip.

Usage

```
cr_import_report_spec(path, validate = TRUE)
```

Arguments

path	Input JSON path.
validate	Validate the reconstructed specification.

cr_interpretation_lookup *Look up explicit interpretation text*

Description

Look up explicit interpretation text

Usage

```
cr_interpretation_lookup(classification, dictionary, default = NA_character_)
```

Arguments

classification	Character vector.
dictionary	Named character vector or named list.
default	Value used for unknown or missing classifications.

cr_lab_report *Assemble a laboratory report without recalculating analytical values*

Description

Assemble a laboratory report without recalculating analytical values

Usage

```
cr_lab_report(
  experiment = NULL,
  spec = cr_report_spec(),
  qc = NULL,
  result_graphic = FALSE,
  include_audit_appendix = NULL,
  strict = TRUE,
  style = NULL,
  profile = NULL
)
```

Arguments

experiment	Optional validated cr_experiment; it is retained by reference semantics of ordinary R lists and never modified.
spec	A cr_report_spec.
qc	Optional concise QC table from cr_report_qc().

result_graphic	Optional precomputed plot, or TRUE to build one from result thresholds when possible.
include_audit_appendix	Include a compact audit appendix in the rendered document.
strict	Apply strict specification validation.
style	A cr_report_style() shared by the document and its embedded report figures. Legacy cr_plot_style objects are converted.
profile	Optional cr_report_profile() containing laboratory defaults, labels, required fields, and organization-wide visual settings.

---

cr_lab_report_from_evidence *Assemble a laboratory report with validated, source-bound quoted values*

Description

Claim identifiers become custom-field names and their displays become field values. Existing custom fields with the same names are rejected. The existing report renderer and audit exporter revalidate source files and bound fields. Other report fields and free text are not verified by this contract.

Usage

```
cr_lab_report_from_evidence(evidence, root, spec = cr_report_spec(), ...)
```

Arguments

evidence	A cr_report_evidence() object.
root	Current source-tree root. Retained for local validation, but excluded from report data and portable evidence hashes.
spec	A cr_report_spec().
...	Other arguments to cr_lab_report().

Value

A cr_lab_report with additional evidence and local root fields.

cr_linetypes *Stable line types for categorical figures*

Description

Stable line types for categorical figures

Usage

```
cr_linetypes(n = NULL, names = NULL)
```

Arguments

n	Number required, at most six.
names	Optional category names.

cr_logistic *Univariate logistic regression of treatment vs. control*

Description

Fits a univariate logistic regression on a single fluorescence channel for the binary outcome `treatment == treatment` (vs. `treatment == control`). Returns a `cr_result` with the fitted model, ROC curve and AUC.

Usage

```
cr_logistic(
  experiment,
  channel,
  treatment,
  control,
  level = c("cell", "replicate")
)
```

Arguments

experiment	A `cr_experiment`.
channel	Channel name.
treatment	Name of treatment group.
control	Name of control group.
level	`"cell"` (default) or `"replicate"`.

Value

A `cr_result`.

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
res <- cr_logistic(exp,
                   channel = "marker_1",
                   treatment = "CompoundA_high",
                   control = "Untreated")
print(res)
```

`cr_macro_name` *Build a macro-safe name*

Description

Turns a label into an identifier that a LaTeX control sequence accepts: letters only, with digits spelled out and every other character dropped. `"CompoundA_5min"` becomes `"CompoundAfivemin"`.

Usage

```
cr_macro_name(x, prefix = NULL)
```

Arguments

`x`	Character vector of labels.
`prefix`	Optional prefix, transliterated the same way.

Value

A character vector of the same length as `x`.

See Also

`cr_macros()`.

Other macros: `cr_enumerate()`, `cr_format_number()`, `cr_macros()`, `cr_macros_from()`

Examples

```
cr_macro_name(c("CompoundA_5min", "interval-2", "n cells"))
cr_macro_name("estimate", prefix = "screen")
```

`cr_macros` *Emit named values as a generated include file*

Description

Writes every number a write-up quotes into one machine-generated file, so the text can reference the value by name instead of repeating the digits. A number that lives in exactly one place cannot disagree with itself: re-run the analysis, re-emit the file, and the document is current. Splicing an estimate from one run beside an interval from another stops being possible.

Usage

```
cr_macros(
  values,
  file,
  format = c("tex", "json", "yaml"),
  prefix = NULL,
  header = NULL,
  digits = 3,
  big_mark = ",",
  na = "--"
)
```

Arguments

`values`	A named list or named atomic vector of scalars. Nested named lists are flattened, joining the names.
`file`	Output path. When `NULL` the formatted lines are returned instead of being written.
`format`	`"tex"` (default) writes `\newcommand` definitions with presentation-formatted values; `"json"` and `"yaml"` write the underlying values, rounded but otherwise raw, for downstream tools.
`prefix`	Optional prefix prepended to every name. A prefix is worth setting for `"tex"` output: it keeps generated names clear of the commands LaTeX already defines, such as `\label` or `\date`, which cannot be redefined with `\newcommand`.
`header`	Optional character vector of header lines. `NULL` emits a default "generated file, do not edit" banner; `character()` emits none.
`digits`	Decimal places used when formatting numbers. Counts — integers, and round values of 1000 or more — are written without decimals and with `big_mark` separators.
`big_mark`	Thousands separator used in `"tex"` output.
`na`	Placeholder for non-finite values in `"tex"` output.

Details

LaTeX control sequences may contain letters only, so names are transliterated by `cr_macro_name()` — `"CompoundA_5min"` becomes `\CompoundAfivemin`. Two source names that transliterate to the same macro are an error rather than a silent overwrite.

Value

The output path (invisibly), or a character vector of lines when `file` is `NULL`.

See Also

`cr_macros_from()` to derive the values from a report, `cr_macro_name()`, `cr_format_number()`, `cr_enumerate()`.

Other macros: `cr_enumerate()`, `cr_format_number()`, `cr_macro_name()`, `cr_macros_from()`

Examples

```
vals <- list(cells_analyzed = 128400L, units_analyzed = 96L,
             top_estimate = 1.42, top_ci_low = 0.55)
cat(cr_macros(vals, file = NULL), sep = "\n")

f <- tempfile(fileext = ".tex")
cr_macros(vals, f, prefix = "screen")
cat(readLines(f)[1:2], sep = "\n")
```

---

cr_macros_from *Derive generated numbers from a report or a results table*

Description

Walks an assembled report (or any table of contrasts) and turns it into the named values `cr_macros()` emits: the disposition counts, one entry per contrast and numeric column, and the enumerations that a sentence would otherwise state by hand — which intervals exclude the null, and how many. Enumerations are the entries most worth generating, because a hand-typed list is wrong as soon as one row of the analysis changes.

Usage

```
cr_macros_from(x, file, label_cols = NULL, ci = c("ci_low", "ci_high"), ...)
```

Arguments

`x`	A `cr_report`, or a data frame with one row per contrast.
`file`	Output path, or `NULL` to return the lines.
`label_cols`	Character vector naming the columns that identify a row. `NULL` (default) picks the first available of `group`, `compound`, `treatment`, `term`, `name`, together with `contrast` or `comparison` when present.

ci	Names of the lower and upper confidence-bound columns, used for the "excludes zero" enumerations. NULL disables them.
...	Passed to cr_macros() (format, prefix, digits, ...).

Value

The output path (invisibly), or a character vector of lines when file is NULL.

See Also

cr_macros().

Other macros: cr_enumerate(), cr_format_number(), cr_macro_name(), cr_macros()

Examples

```
eff <- data.frame(
  group = c("CompoundA", "CompoundB", "CompoundC"),
  estimate = c(1.42, 0.31, -0.88),
  ci_low = c(0.55, -0.10, -1.60),
  ci_high = c(2.29, 0.72, -0.16)
)
cat(cr_macros_from(eff, file = NULL), sep = "\n")
```

---

cr_marker_rules *Declare how parenthetical file-name markers are interpreted*

Description

Acquisition software and operators annotate exports with short markers, and those markers change the analysis. A cr_marker_rules object states which regular expression means what, so that a marker becomes a typed flag instead of a string somebody has to remember.

Usage

```
cr_marker_rules(
  merge_unit = NULL,
  partial_plate = NULL,
  omitted_reagent = NULL,
  reacquisition = NULL,
  lot = NULL,
  capture_unknown = TRUE,
  ignore_case = TRUE
)
```

Arguments

`merge_unit`	Regular expression marking a file as one half of a two-pass acquisition, or `NULL`.
`partial_plate`	Regular expression matched against the container (directory) name, or `NULL`.
`omitted_reagent`	Regular expression marking the specificity arm, or `NULL`.
`reacquisition`	Regular expression marking a repeated read, or `NULL`.
`lot`	Regular expression marking a reagent lot, or `NULL`.
`capture_unknown`	Logical. Capture an unmatched trailing parenthetical into `variant`. Default `TRUE`.
`ignore_case`	Logical. Match case-insensitively. Default `TRUE`.

Details

The distinctions the rules encode are deliberate:

- `merge_unit` sits on a **file** and means one spatial unit was acquired in two passes. The two files are one unit and must be merged before any per-unit balancing, or the unit contributes twice the cells of its neighbours.
- `partial_plate` sits on a **container** (the plate directory) and means a partly filled plate. It has no downstream consequence. Collapsing the two into one flag hides the one that matters, so they are matched against different strings.
- `omitted_reagent` marks the specificity arm, where the detection reagent was left out. These acquisitions are never samples.
- `reacquisition` marks a repeated read of a unit that already has a plain sibling file.
- `lot` marks a different reagent lot. A lot marker is *not* an omitted-reagent control; pooling the two inverts the meaning of the arm.

Any trailing parenthetical that matches none of the rules is captured verbatim into a `variant` column when `capture_unknown = TRUE`, rather than being guessed at or silently dropped.

Value

An object of class `cr_marker_rules` (a list).

See Also

`cr_extract_markers()`, `cr_parse_paths()`, `cr_merge_rules()`.

Other import: `cr_assign_units()`, `cr_centroid_overlap()`, `cr_column_map()`, `cr_extract_markers()`, `cr_filename_grammar()`, `cr_merge_rules()`, `cr_parse_paths()`, `cr_path_spec()`, `cr_read_cellprofiler()`, `cr_read_cells()`, `cr_read_design()`, `cr_read_export()`, `cr_read_exports()`, `cr_read_qupath()`, `cr_read_segmantr()`, `cr_unit_map()`

Examples

```
rules <- cr_marker_rules(
  merge_unit = "\\(split\\)",
  partial_plate = "\\(partial\\)",
  omitted_reagent = "\\(no reagent\\)",
  reacquisition = "\\(repeat\\)",
  lot = "\\(lot[A-Z]\\)"
)
rules
```

---

cr_merge_experiments *Merge multiple experiments*

Description

Combines two or more `cr_experiment` objects that share the same schema (channel panel, spatial unit). Useful for multi-plate studies. Well/slide names must be unique across inputs (prefix them with a plate identifier if necessary).

Usage

```
cr_merge_experiments(...)
```

Arguments

`...`	`cr_experiment` objects.

Value

A single merged `cr_experiment`.

Examples

```
e1 <- cr_example_experiment(seed = 1, n_cells_per_well = 10)
e2 <- cr_example_experiment(seed = 2, n_cells_per_well = 10)
e2$cells$well <- paste0("p2_", e2$cells$well)
e2$design$well <- paste0("p2_", e2$design$well)
merged <- cr_merge_experiments(e1, e2)
```

`cr_merge_rules` *Declare how files are merged into analysis units*

Description

States which replicate suffixes and marker flags mean "these files are one unit" and which mean "these files are different units". Getting this wrong is silent in both directions, so each rule is explicit.

Usage

```
cr_merge_rules(
  merge_suffix = "\\.1$",
  merge_marker = "merge_unit",
  keep_separate = "\\.2$",
  reacquisition = "reacquisition",
  merge_reacquisition = TRUE,
  separate_suffix = "re"
)

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

Arguments

`merge_suffix`	Regular expression matched against the replicate index, marking a second acquisition pass. `NULL` disables.
`merge_marker`	Name of a logical column that must be `TRUE` for a `merge_suffix` match to merge, or `NULL` to merge on the suffix alone. When the column is absent no suffix merging happens.
`keep_separate`	Regular expression matched against the replicate index, marking indices that must never merge. `NULL` disables.
`reacquisition`	Name of a logical column marking repeated reads, or `NULL`.
`merge_reacquisition`	Logical. Fold repeated reads into their plain sibling. Default `TRUE`.
`separate_suffix`	Suffix appended to the replicate index of a repeated read when `merge_reacquisition = FALSE`. Default `"re"`.
`x`	A `cr_merge_rules`.
`...`	Ignored.

Details

- `merge_suffix` matches the replicate index of a file that is the *second pass* over a unit that already exists. Left unmerged, such a unit draws a full cell allocation twice and is double-weighted in its own batch.

- merge_marker names a logical column (typically produced by cr_extract_markers()) that must also be TRUE before a suffix match is merged. Requiring the marker is the conservative default: a suffix on its own is not evidence.
- keep_separate matches replicate indices that look mergeable but are a different physical unit. It always wins over merge_suffix. cr_centroid_overlap() is the evidence test for deciding which of the two a given suffix is.
- reacquisition names the logical column marking a repeated read. With merge_reacquisition = TRUE such a file folds into its plain sibling; otherwise separate_suffix is appended so it stays a unit of its own.

Value

An object of class cr_merge_rules (a list).

x, invisibly.

See Also

cr_assign_units(), cr_extract_markers(), cr_centroid_overlap().

Other import: cr_assign_units(), cr_centroid_overlap(), cr_column_map(), cr_extract_markers(), cr_filename_grammar(), cr_marker_rules(), cr_parse_paths(), cr_path_spec(), cr_read_cellprofiler(), cr_read_cells(), cr_read_design(), cr_read_export(), cr_read_exports(), cr_read_qupath(), cr_read_segmantr(), cr_unit_map()

Examples

```
cr_merge_rules()
cr_merge_rules(merge_marker = NULL, keep_separate = NULL)
```

---

cr_n_cells *Count cells in a* cr_experiment

Description

Count cells in a cr_experiment

Usage

```
cr_n_cells(experiment, by = NULL)
```

Arguments

experiment	A cr_experiment.
by	Optional character vector of grouping columns from the design or cells table. If NULL, returns total count.

Value

Either a scalar integer (no grouping) or a tibble with cell counts per group.

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 10)
cr_n_cells(exp)
cr_n_cells(exp, by = "treatment")
```

---

cr_normalize *Normalize intensity data*

Description

Applies a channel-wise normalization. The normalized channel values replace the original column in `cells`.

Usage

```
cr_normalize(
  experiment,
  channel,
  method = c("background", "control", "zscore", "robust_zscore", "quantile"),
  control_group = NULL,
  ...
)
```

Arguments

`experiment`	A `cr_experiment`.
`channel`	Channel column to normalize.
`method`	One of `"background"`, `"control"`, `"zscore"`, `"robust_zscore"`, `"quantile"`.
`control_group`	For `method = "control"`: the treatment name (in `design$treatment`) whose median / mean is used as reference.
`...`	Additional arguments passed to the underlying method.

Details

- `"background"` subtracts per-well background via `cr_background_subtract()` (percentile method, 5th percentile).
- `"control"` divides each cell's intensity by the median intensity of cells assigned to `control_group`, then takes the log2 ratio.
- `"zscore"` applies a global Z-score across all cells.
- `"robust_zscore"` uses the median and MAD.

- "quantile" maps the per-well empirical distributions onto the global quantile distribution.

All five methods use a *single, pooled* reference for the whole experiment and overwrite the channel in place. When the reference has to be the control of each cell's own batch – the usual situation once several plates, runs or acquisition days are involved – use cr_standardize_batch() instead, which adds new columns rather than overwriting the measured signal.

Value

A modified cr_experiment.

See Also

cr_standardize_batch() for per-batch standardization, cr_background_subtract(), cr_correct_batch().

Other normalization functions: cr_background_subtract()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
cr_normalize(exp, channel = "marker_1", method = "robust_zscore")
```

---

cr_palette *Colour-vision-safe palette*

Description

Returns hex colours from the palette every cr_plot_* function uses for discrete groupings. The qualitative set is Okabe-Ito, the standard colour-vision-safe qualitative palette; it deliberately contains no red/green pair, so hue never encodes an opposition that a large minority of readers cannot separate.

Usage

```
cr_palette(
  n = NULL,
  type = c("qualitative", "sequential", "diverging"),
  names = NULL,
  mode = c("colour", "grayscale")
)
```

Arguments

n	Number of colours to return. NULL (default) returns the whole palette: 8 colours for "qualitative", 256 for the continuous types. Requesting more than 8 qualitative colours interpolates between the anchors and warns, because the result is no longer guaranteed to be colour-vision-safe; prefer facetting over more than 8 hues.

type	One of "qualitative" (discrete groups), "sequential" (one-directional magnitude) or "diverging" (signed magnitude around a meaningful zero, ramped blue-to-vermillion rather than blue-to-red-green).
names	Optional character vector of names to attach to the returned colours, for use as the values argument of a manual scale. Must have the same length as the result.
mode	"colour" or a monotonically separated "grayscale" palette.

Details

Hue alone is not treated as a sufficient encoding: use cr_shapes() (or cr_scale_group(), which wires both at once) so that every grouping is carried by shape as well as colour.

Value

A character vector of hex colour strings, named when names is supplied or when the whole qualitative palette is returned.

See Also

cr_shapes(), cr_scale_group(), cr_theme()

Other plot design: cr_scale_group(), cr_shapes(), cr_theme()

Examples

```
cr_palette()
cr_palette(3)
cr_palette(5, names = c("a", "b", "c", "d", "e"))
cr_palette(7, type = "diverging")
```

---

cr_parse_paths *Parse design facts out of export paths*

Description

Turns a vector of export paths into one row of design information per file: the directory levels below root, the file-name markers, the replicate index and the grammar tokens.

Usage

```
cr_parse_paths(
  paths,
  root = NULL,
  levels = NULL,
  grammar = NULL,
  markers = NULL,
  strict = TRUE,
  spec = NULL,
  call = rlang::caller_env()
)
```

Arguments

paths	Character vector of file paths.
root	Directory the paths sit under. Required when levels is given.
levels	Directory level specification; see cr_path_spec().
grammar	A cr_filename_grammar(), or NULL.
markers	A cr_marker_rules() object, or NULL.
strict	Logical. Abort on an unparseable file name. When spec is supplied and strict is not, the spec's setting is used.
spec	Optional cr_path_spec() supplying any of levels, grammar, markers and strict that are not given directly.
call	The execution environment of the calling function. Used for error reporting; experts only.

Details

With strict = TRUE (the default) a file name that matches no core_patterns entry of the grammar aborts the parse. That is the intended behaviour: a name that parses to defaults instead of failing loudly can move a treated unit into a control arm without anything in the analysis noticing.

Value

A tibble with one row per path: source_file, source_path, one column per directory level, the marker flag columns, variant, core, replicate, one column per grammar token, and the parse_ok / parse_error outcome.

See Also

cr_path_spec(), cr_filename_grammar(), cr_marker_rules(), cr_read_exports().

Other import: cr_assign_units(), cr_centroid_overlap(), cr_column_map(), cr_extract_markers(), cr_filename_grammar(), cr_marker_rules(), cr_merge_rules(), cr_path_spec(), cr_read_cellprofiler(), cr_read_cells(), cr_read_design(), cr_read_export(), cr_read_exports(), cr_read_qupath(), cr_read_segmantr(), cr_unit_map()

Examples

```
root <- file.path(tempdir(), "cr_paths_demo")
paths <- file.path(
  root, "Run1", "CompoundA", "Plate_1",
  c("CompoundA_vehicle_1.csv", "CompoundA_5min_10uM_treated_1.csv")
)
cr_parse_paths(
  paths,
  root = root,
  levels = c("run", "compound", "plate"),
  grammar = cr_filename_grammar(
    tokens = list(interval = "[0-9]+min", dose = "[0-9]+uM"),
    defaults = list(interval = "none", dose = "vehicle"),
```

```
    prefix_strip = "CompoundA"
  )
)
```

`cr_path_spec` *Bundle a directory and file-name specification*

Description

A `cr_path_spec` holds everything needed to recover design facts from where a file sits and what it is called: the meaning of each directory level, the file-name grammar, the marker rules and whether an unparseable name is an error.

Usage

```
cr_path_spec(levels = NULL, grammar = NULL, markers = NULL, strict = TRUE)

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

Arguments

`levels`	Either an unnamed character vector naming the directory levels below the root in order (use `NA` or `""` to skip a level), or a named integer vector mapping column names to level indices, where negative indices count back from the file (`-1` is the directory containing the file).
`grammar`	A `cr_filename_grammar()`, or `NULL`.
`markers`	A `cr_marker_rules()` object, or `NULL`.
`strict`	Logical. Treat a file name that does not match the grammar as an error. Default `TRUE`.
`x`	A `cr_path_spec`.
`...`	Ignored.

Value

An object of class `cr_path_spec` (a list).

`x`, invisibly.

See Also

`cr_parse_paths()`, `cr_read_exports()`.

Other import: `cr_assign_units()`, `cr_centroid_overlap()`, `cr_column_map()`, `cr_extract_markers()`, `cr_filename_grammar()`, `cr_marker_rules()`, `cr_merge_rules()`, `cr_parse_paths()`, `cr_read_cellprofiler()`, `cr_read_cells()`, `cr_read_design()`, `cr_read_export()`, `cr_read_exports()`, `cr_read_qupath()`, `cr_read_segmantr()`, `cr_unit_map()`

Examples

```
spec <- cr_path_spec(
  levels = c(run = 1L, compound = 2L, plate = -1L),
  grammar = cr_filename_grammar(
    tokens = list(interval = "[0-9]+min", dose = "[0-9]+uM")
  ),
  markers = cr_marker_rules(merge_unit = "\\(split\\)")
)
spec
```

`cr_plot_comparison` *Comparison panel (box, fold change, p-value) for a single result*

Description

Comparison panel (box, fold change, p-value) for a single result

Usage

```
cr_plot_comparison(result, experiment)
```

Arguments

`result`	A `cr_result`.
`experiment`	A `cr_experiment` (required to reconstruct the underlying intensity data).

Value

A `ggplot` object.

See Also

Other visualisation: `cr_plot_dose_response()`, `cr_plot_effect_sizes()`, `cr_plot_foldchange()`, `cr_plot_heatmap()`, `cr_plot_histogram()`, `cr_plot_intensity()`, `cr_plot_plate()`, `cr_plot_qc()`, `cr_plot_roc()`, `cr_plot_scatter()`, `cr_plot_spatial()`, `cr_plot_timeline()`

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
res <- cr_test(exp, "marker_1", "CompoundA_high", "Untreated",
               test = "mann_whitney", level = "replicate")
cr_plot_comparison(res, exp)
```

cr_plot_dose_response *Dose-response plot*

Description

Dose-response plot

Usage

```
cr_plot_dose_response(fit)
```

Arguments

fit	A cr_dose_response.

Value

A ggplot object.

See Also

Other visualisation: cr_plot_comparison(), cr_plot_effect_sizes(), cr_plot_foldchange(), cr_plot_heatmap(), cr_plot_histogram(), cr_plot_intensity(), cr_plot_plate(), cr_plot_qc(), cr_plot_roc(), cr_plot_scatter(), cr_plot_spatial(), cr_plot_timeline()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
exp$design$dose <- ifelse(exp$design$treatment == "CompoundA_high",
                          500, exp$design$dose)
fit <- cr_dose_response(exp, channel = "marker_1", model = "4pl")
cr_plot_dose_response(fit)
```

cr_plot_effect_sizes *Forest plot of effect sizes*

Description

Convenience wrapper around cr_plot_forest() for the result objects produced by cr_test() and cr_test_all(). Use cr_plot_forest() directly for an effect-size table with its own labelling, facetting or colouring.

Usage

```
cr_plot_effect_sizes(results, method = "cohens_d", ...)
```

Arguments

results	A single cr_result, a list of cr_results (from cr_test_all()) or a precomputed tibble with columns treatment, method, estimate, ci_low, ci_high.
method	Effect-size method to plot (default "cohens_d").
...	Further arguments passed to cr_plot_forest(), for example facet_by, colour_by or descending.

Value

A ggplot object.

See Also

cr_plot_forest()

Other visualisation: cr_plot_comparison(), cr_plot_dose_response(), cr_plot_foldchange(), cr_plot_heatmap(), cr_plot_histogram(), cr_plot_intensity(), cr_plot_plate(), cr_plot_qc(), cr_plot_roc(), cr_plot_scatter(), cr_plot_spatial(), cr_plot_timeline()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 20)
all_res <- cr_test_all(exp, "marker_1", "Untreated", level = "replicate")
cr_plot_effect_sizes(all_res)
```

---

cr_plot_estimates *Plot estimates and confidence intervals*

Description

Plot estimates and confidence intervals

Usage

```
cr_plot_estimates(
  data,
  estimate = "estimate",
  lower = "conf_low",
  upper = "conf_high",
  label = "group",
  group = NULL,
  reference = 0,
  order_by = c("design", "estimate", "name"),
  colour_mode = c("colour", "grayscale"),
  style = NULL
)
```

Arguments

`data`	Data frame of finalized estimates.
`estimate`, `lower`, `upper`, `label`	Column names.
`group`	Optional grouping column, redundantly encoded by colour and shape.
`reference`	Optional explicit reference value.
`order_by`	`design`, `estimate`, or `name`.
`colour_mode`	Colour mode.
`style`	Optional `cr_plot_style`.

`cr_plot_foldchange` *Fold-change forest plot*

Description

Fold-change forest plot

Usage

```
cr_plot_foldchange(result)
```

Arguments

`result`	A `cr_result`, a list of `cr_results` (from `cr_test_all()`), or a precomputed data frame with columns `treatment`, `median_log2_fc`.

Value

A `ggplot` object.

See Also

`cr_plot_forest()` for effect sizes with confidence intervals.

Other visualisation: `cr_plot_comparison()`, `cr_plot_dose_response()`, `cr_plot_effect_sizes()`, `cr_plot_heatmap()`, `cr_plot_histogram()`, `cr_plot_intensity()`, `cr_plot_plate()`, `cr_plot_qc()`, `cr_plot_roc()`, `cr_plot_scatter()`, `cr_plot_spatial()`, `cr_plot_timeline()`

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 20)
res <- cr_test_all(exp, "marker_1", "Untreated", level = "replicate")
cr_plot_foldchange(res)
```

`cr_plot_forest` *Forest plot of effect sizes with confidence intervals*

Description

Draws one row per contrast: the point estimate with its confidence interval, a reference line at the null, and the rows ordered by the estimate rather than alphabetically, so the ordering is computed from the data and cannot go stale.

Usage

```
cr_plot_forest(
  effects,
  estimate = "estimate",
  ci_low = "ci_low",
  ci_high = "ci_high",
  label = NULL,
  facet_by = NULL,
  colour_by = NULL,
  method = "cohens_d",
  reference = 0,
  order_by_estimate = TRUE,
  descending = FALSE,
  title = NULL,
  subtitle = NULL,
  x_lab = NULL,
  y_lab = NA
)
```

Arguments

`effects`	A data frame of effect sizes (for example the output of an effect-size grid), a `cr_result`, or a list of `cr_results`.
`estimate, ci_low, ci_high`	Names of the estimate and interval columns.
`label`	Name of the column identifying each row. `NULL` (default) picks the first of `group`, `compound`, `treatment`, `contrast`, `label` or `term` that is present.
`facet_by`	One or two column names to facet on, or `NULL`.
`colour_by`	Column mapped to colour, fill and shape, or `NULL`.
`method`	Effect-size method to keep when `effects` carries a `method` column with several methods (default `"cohens_d"`).
`reference`	Position of the null reference line (default `0`).
`order_by_estimate`	Order rows by the estimate (default `TRUE`). `FALSE` keeps the order of `label`.

descending
: Order largest estimate at the top (default FALSE, which puts the most negative - typically the strongest protection - at the top).

title, subtitle, x_lab, y_lab
: Plot labels. NULL uses a computed default; NA omits the label.

Details

The default subtitle states which intervals exclude the reference. It is computed from the interval bounds for the same reason: the equivalent sentence written by hand in the source pipeline stopped matching the data the first time the data changed.

Value

A ggplot object.

See Also

cr_plot_screen(), cr_plot_effect_sizes()

Other screen figures: cr_plot_qc_gate(), cr_plot_sample_size(), cr_plot_screen(), cr_plot_specificity(), cr_save_plot()

Examples

```
effects <- data.frame(
  group = c("CompoundA", "CompoundB", "CompoundC"),
  estimate = c(-1.2, -0.35, 0.1),
  ci_low = c(-1.9, -0.95, -0.4),
  ci_high = c(-0.5, 0.25, 0.6)
)
cr_plot_forest(effects)
```

---

cr_plot_heatmap *Heatmap of channel medians across groups*

Description

Heatmap of channel medians across groups

Usage

```
cr_plot_heatmap(
  experiment,
  channels,
  group_by = "treatment",
  scale = c("none", "row", "column")
)
```

Arguments

experiment	A cr_experiment.
channels	Character vector of channel names.
group_by	Design column defining rows. Default "treatment".
scale	One of "none", "row", "column". Scaling centres the values, so the fill switches from the sequential to the diverging palette.

Value

A ggplot object.

See Also

Other visualisation: cr_plot_comparison(), cr_plot_dose_response(), cr_plot_effect_sizes(), cr_plot_foldchange(), cr_plot_histogram(), cr_plot_intensity(), cr_plot_plate(), cr_plot_qc(), cr_plot_roc(), cr_plot_scatter(), cr_plot_spatial(), cr_plot_timeline()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
cr_plot_heatmap(exp, c("DAPI", "marker_1", "marker_2"))
```

---

cr_plot_histogram *Histogram of channel intensity*

Description

Histogram of channel intensity

Usage

```
cr_plot_histogram(
  experiment,
  channel,
  group_by = "treatment",
  facet_by = NULL,
  log_x = TRUE
)
```

Arguments

experiment	A cr_experiment.
channel	Channel name.
group_by	Design column used for colour.
facet_by	Design column used for facetting (or NULL).
log_x	Log-transform the x-axis.

Value

A ggplot object.

See Also

Other visualisation: cr_plot_comparison(), cr_plot_dose_response(), cr_plot_effect_sizes(), cr_plot_foldchange(), cr_plot_heatmap(), cr_plot_intensity(), cr_plot_plate(), cr_plot_qc(), cr_plot_roc(), cr_plot_scatter(), cr_plot_spatial(), cr_plot_timeline()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
cr_plot_histogram(exp, "marker_1")
```

---

cr_plot_intensity *Intensity distributions by group*

Description

Intensity distributions by group

Usage

```
cr_plot_intensity(
  experiment,
  channel,
  group_by = "treatment",
  geom = c("violin", "boxplot", "both"),
  log_y = TRUE
)
```

Arguments

experiment	A cr_experiment.
channel	Channel name.
group_by	Design column to colour / facet on. Default "treatment".
geom	"violin", "boxplot" or "both".
log_y	Log-transform the y-axis (default TRUE).

Value

A ggplot object.

See Also

cr_plot_screen() to overlay the unit of replication.

Other visualisation: cr_plot_comparison(), cr_plot_dose_response(), cr_plot_effect_sizes(), cr_plot_foldchange(), cr_plot_heatmap(), cr_plot_histogram(), cr_plot_plate(), cr_plot_qc(), cr_plot_roc(), cr_plot_scatter(), cr_plot_spatial(), cr_plot_timeline()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
cr_plot_intensity(exp, channel = "marker_1")
```

---

cr_plot_plate *Plate-layout heatmap*

Description

Draws a plate-layout heatmap for a 96 or 384-well plate, showing a summary metric (median intensity, cell count, etc.) per well.

Usage

```
cr_plot_plate(
  experiment,
  channel,
  metric = c("median", "mean", "cv", "n_cells")
)
```

Arguments

experiment	A cr_experiment.
channel	Channel name for the metric computation.
metric	One of "median", "mean", "cv", "n_cells".

Value

A ggplot object.

See Also

cr_theme(), cr_palette()

Other visualisation: cr_plot_comparison(), cr_plot_dose_response(), cr_plot_effect_sizes(), cr_plot_foldchange(), cr_plot_heatmap(), cr_plot_histogram(), cr_plot_intensity(), cr_plot_qc(), cr_plot_roc(), cr_plot_scatter(), cr_plot_spatial(), cr_plot_timeline()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
cr_plot_plate(exp, channel = "marker_1", metric = "median")
```

cr_plot_qc *QC dashboard*

Description

Combines cell count per well, area distributions and intensity distributions into a single multi-panel figure.

Usage

```
cr_plot_qc(experiment, channel = NULL)
```

Arguments

experiment	A cr_experiment.
channel	Channel name used for the intensity panel.

Value

A ggplot object (facetted).

See Also

cr_plot_qc_gate() for the gate against each unit's own control.

Other visualisation: cr_plot_comparison(), cr_plot_dose_response(), cr_plot_effect_sizes(), cr_plot_foldchange(), cr_plot_heatmap(), cr_plot_histogram(), cr_plot_intensity(), cr_plot_plate(), cr_plot_roc(), cr_plot_scatter(), cr_plot_spatial(), cr_plot_timeline()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
cr_plot_qc(exp, channel = "marker_1")
```

cr_plot_qc_gate *Quality-control gate diagnostic plot*

Description

Plots each unit's statistic against the statistic of the control it is gated on, with the line of equality. Points above the line pass a "greater" gate; points below it fail and are excluded.

Usage

```
cr_plot_qc_gate(
  gate,
  statistic = NULL,
  reference = NULL,
  fails_median = NULL,
  fails_mean = NULL,
  direction = c("greater", "less"),
  log_scale = TRUE,
  label = NULL,
  title = NULL,
  subtitle = NULL,
  x_lab = NULL,
  y_lab = NULL
)
```

Arguments

gate	A cr_qc_gate object, a list carrying a units data frame, or a data frame with one row per unit.
statistic	Name of the unit statistic column. NULL (default) auto-detects.
reference	Name of the control statistic column. NULL (default) auto-detects.
fails_median, fails_mean	Names of the logical verdict columns under the two centres. NULL (default) auto-detects; when neither is present the verdict is computed from statistic, reference and direction.
direction	"greater" (default) when a unit must exceed its control to pass, "less" when it must fall below.
log_scale	Draw both axes on a log scale when every value is positive (default TRUE).
label	Optional column whose values annotate the failing units.
title, subtitle, x_lab, y_lab	Plot labels. NULL uses a computed default; NA omits the label.

Details

Where the input carries a verdict under both centres, median and mean, the units whose verdict *depends* on which centre was used are drawn as their own class. That distinction matters because a right-skewed signal has a mean above its median, so gating a raw median against a control mean is silently stricter than the stated rule, and a gate that can only drop low units manufactures apparent effects when it is too strict.

Value

A ggplot object.

See Also

cr_plot_qc() for the distribution dashboard.

Other screen figures: cr_plot_forest(), cr_plot_sample_size(), cr_plot_screen(), cr_plot_specificity(), cr_save_plot()

Examples

```
set.seed(1)
units <- data.frame(
  well_id = paste0("W", 1:24),
  unit_median = c(rlnorm(20, 5, 0.3), rlnorm(4, 3.9, 0.2)),
  ctrl_median = rlnorm(24, 4.2, 0.1)
)
cr_plot_qc_gate(units, label = "well_id")
```

---

cr_plot_qc_summary *Compact text-first QC status plot*

Description

Compact text-first QC status plot

Usage

```
cr_plot_qc_summary(qc, mode = c("colour", "grayscale"))
```

Arguments

qc	A report QC table.
mode	Colour mode.

---

cr_plot_result_position *Plot a value relative to caller-supplied boundaries*

Description

Plot a value relative to caller-supplied boundaries

Usage

```
cr_plot_result_position(
  value,
  thresholds,
  labels = NULL,
  xlim = NULL,
  unit = NULL,
  classification = NULL,
  show_value = TRUE,
  mode = c("colour", "grayscale"),
  style = NULL
)
```

Arguments

value	Single finite numeric value.
thresholds	One or more sorted finite boundaries.
labels	Optional labels for the resulting intervals; must have one more item than thresholds.
xlim	Optional two-value display range.
unit, classification	Optional user-supplied marker labels.
show_value	Include the numeric value below the marker.
mode	Colour or grayscale rendering.
style	Optional cr_plot_style().

---

cr_plot_roc *ROC curve plot*

Description

ROC curve plot

Usage

```
cr_plot_roc(result)
```

Arguments

result	A single cr_result or a list of such results.

Value

A ggplot object.

See Also

Other visualisation: cr_plot_comparison(), cr_plot_dose_response(), cr_plot_effect_sizes(), cr_plot_foldchange(), cr_plot_heatmap(), cr_plot_histogram(), cr_plot_intensity(), cr_plot_plate(), cr_plot_qc(), cr_plot_scatter(), cr_plot_spatial(), cr_plot_timeline()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
res <- cr_logistic(exp, "marker_1", "CompoundA_high", "Untreated")
cr_plot_roc(res)
```

---

cr_plot_sample_size *Sample-size comparison plot*

Description

Draws, per group, the number of units a confirmatory study would need when powered on the observed effect beside the number needed when powered on the confidence bound nearer the null.

Usage

```
cr_plot_sample_size(
  sizes,
  label = NULL,
  observed = "n_observed",
  conservative = "n_conservative",
  available = NULL,
  colour_by = NULL,
  log_y = TRUE,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_lab = NA,
  y_lab = NULL
)
```

Arguments

sizes	A data frame with one row per group, carrying the two sample-size columns.
label	Name of the grouping column. NULL (default) picks the first of group, compound, treatment or label that is present.
observed, conservative	Names of the two sample-size columns.
available	Optional name of a column giving the units already available per group; folded into the axis labels in brackets.
colour_by	Column mapped to fill, or NULL for a single colour.

`log_y` Draw the count axis on a log scale (default `TRUE`, because the two bars routinely differ by an order of magnitude).

`title`, `subtitle`, `caption`, `x_lab`, `y_lab`
Plot labels. `NULL` uses a computed default; `NA` omits the label.

Details

The gap between the two bars is the point of the figure. Powering on the observed effect of a screen's top hit is circular - that group is the largest only because it was selected for being largest - so the conservative bar is the one a confirmatory design should be built on, and a conservative figure exists only where the interval excludes the null.

Value labels are drawn in ink above the bars, never inside them, and the number of units already available is folded into the axis label rather than plotted as a marker that collides with the labels on short bars.

Value

A `ggplot` object.

See Also

`cr_plot_forest()`

Other screen figures: `cr_plot_forest()`, `cr_plot_qc_gate()`, `cr_plot_screen()`, `cr_plot_specificity()`, `cr_save_plot()`

Examples

```
sizes <- data.frame(
  group = c("CompoundA", "CompoundB", "CompoundC"),
  n_observed = c(12, 84, 640),
  n_conservative = c(46, NA, NA),
  n_available = c(6, 6, 6)
)
cr_plot_sample_size(sizes, available = "n_available")
```

---

cr_plot_scatter *Biaxial scatter plot of two channels*

Description

Biaxial scatter plot of two channels

Usage

```
cr_plot_scatter(
  experiment,
  channel_x,
  channel_y,
  color_by = "treatment",
  log_x = TRUE,
  log_y = TRUE
)
```

Arguments

experiment	A cr_experiment.
channel_x	X-axis channel.
channel_y	Y-axis channel.
color_by	Design column used for colour. Default "treatment".
log_x, log_y	Log-transform the respective axis.

Value

A ggplot object.

See Also

Other visualisation: cr_plot_comparison(), cr_plot_dose_response(), cr_plot_effect_sizes(), cr_plot_foldchange(), cr_plot_heatmap(), cr_plot_histogram(), cr_plot_intensity(), cr_plot_plate(), cr_plot_qc(), cr_plot_roc(), cr_plot_spatial(), cr_plot_timeline()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
cr_plot_scatter(exp, "DAPI", "marker_1")
```

---

cr_plot_screen *Distribution figure with the unit of replication overlaid*

Description

Draws width-scaled violins of the cell-level value for each group, with the per-unit means overplotted as jittered points. Cells are the observations; the unit (well, slide or merged acquisition) is the unit of replication, and drawing both in one panel is what keeps a reader from reading cell counts as replicate counts.

Usage

```
cr_plot_screen(
  cells,
  value = "log2_fc",
  group_var = "treatment",
  units = NULL,
  unit_value = NULL,
  facet_by = NULL,
  colour_by = NULL,
  reference = 0,
  seed = NULL,
  title = NULL,
  subtitle = NULL,
  x_lab = NA,
  y_lab = NULL
)
```

Arguments

`cells`	Cell-level data: a data frame, or a `cr_experiment` whose cells are joined to its design.
`value`	Name of the cell-level value column, typically a standardised value such as a log fold change (default `"log2_fc"`).
`group_var`	Name of the column defining the x-axis groups (default `"treatment"`).
`units`	Unit-level data. Either a data frame of one row per unit, or the name of a column in `cells` identifying the unit, in which case the per-unit means are computed. `NULL` draws the violins alone.
`unit_value`	Name of the unit-level value column. Defaults to `value`.
`facet_by`	One or two column names to facet on, or `NULL`.
`colour_by`	Column mapped to fill and shape. `NULL` (default) draws a single colour, because hue that encodes nothing is decoration.
`reference`	Horizontal reference line, or `NULL` for none (default `0`, the null of a log fold change).
`seed`	Integer seed for the point jitter, so the figure reproduces.
`title`, `subtitle`, `x_lab`, `y_lab`	Plot labels. `NULL` uses a computed default; `NA` omits the label.

Details

The point layer is drawn with a fixed jitter seed so the figure is reproducible, and the subtitle states which layer is the unit of replication rather than leaving it to the caption.

Value

A `ggplot` object.

See Also

cr_plot_forest(), cr_theme(), cr_scale_group()

Other screen figures: cr_plot_forest(), cr_plot_qc_gate(), cr_plot_sample_size(), cr_plot_specificity(), cr_save_plot()

Examples

```
set.seed(1)
cells <- data.frame(
  treatment = rep(c("Vehicle", "CompoundA", "CompoundB"), each = 120),
  well_id = rep(paste0("W", 1:18), each = 20),
  log2_fc = c(rnorm(120, 0, 0.5), rnorm(120, -0.8, 0.5),
              rnorm(120, -0.2, 0.5))
)
cr_plot_screen(cells, group_var = "treatment", units = "well_id", seed = 1)
```

---

cr_plot_spatial *Spatial scatter of cells in a well*

Description

Spatial scatter of cells in a well

Usage

```
cr_plot_spatial(experiment, channel, well = NULL)
```

Arguments

experiment	A cr_experiment.
channel	Channel used for colouring points.
well	Spatial unit (well or slide ID) to plot. If NULL, the first well is chosen.

Value

A ggplot object.

See Also

Other visualisation: cr_plot_comparison(), cr_plot_dose_response(), cr_plot_effect_sizes(), cr_plot_foldchange(), cr_plot_heatmap(), cr_plot_histogram(), cr_plot_intensity(), cr_plot_plate(), cr_plot_qc(), cr_plot_roc(), cr_plot_scatter(), cr_plot_timeline()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
cr_plot_spatial(exp, "marker_1")
```

`cr_plot_specificity` *Specificity control plot*

Description

Draws the arms of the specificity control side by side: the assay with the detection reagent present against the same exposure with the reagent omitted. No amount of batch standardisation can show that a readout is reagent-dependent rather than background autofluorescence, which is what makes this the strongest validation a method-establishment result has.

Usage

```
cr_plot_specificity(
  spec,
  arm = "arm",
  value = NULL,
  arm_levels = NULL,
  colour_by = NULL,
  log_y = TRUE,
  ratio_arms = NULL,
  title = NULL,
  subtitle = NULL,
  x_lab = NA,
  y_lab = NULL
)
```

Arguments

`spec`	Cell-level or per-arm data: a data frame, or a list carrying one in a `spec` or `specificity` element.
`arm`	Name of the column identifying the arm (default `"arm"`).
`value`	Name of the value column. `NULL` (default) auto-detects: `median_signal` for a summarised table, otherwise `value`.
`arm_levels`	Optional character vector fixing the order of the arms.
`colour_by`	Column mapped to fill, or `NULL` to fill by arm.
`log_y`	Draw the signal axis on a log scale (default `TRUE`).
`ratio_arms`	Optional length-2 character vector naming the arms whose median ratio is reported in the subtitle, as `c(numerator, denominator)`. When `spec` carries a `signal_to_background` attribute that value is used instead.
`title`, `subtitle`, `x_lab`, `y_lab`	Plot labels. `NULL` uses a computed default; `NA` omits the label.

Details

Accepts either cell-level data (one row per cell, drawn as violins with the arm medians annotated) or an already-summarised table (one row per arm, drawn as columns).

Value

A ggplot object.

See Also

Other screen figures: cr_plot_forest(), cr_plot_qc_gate(), cr_plot_sample_size(), cr_plot_screen(), cr_save_plot()

Examples

```
set.seed(1)
spec <- data.frame(
  arm = rep(c("reagent omitted\n+ vehicle", "reagent omitted\n+ exposed",
              "reagent present\n+ vehicle", "reagent present\n+ exposed"),
            each = 60),
  value = c(rlnorm(60, 3, 0.3), rlnorm(60, 3.05, 0.3),
            rlnorm(60, 4.5, 0.3), rlnorm(60, 6.2, 0.3))
)
cr_plot_specificity(spec)
```

---

cr_plot_style *Construct a reusable plot style*

Description

Construct a reusable plot style

Usage

```
cr_plot_style(
  mode = c("colour", "grayscale"),
  variant = c("publication", "report"),
  base_size = 9,
  base_family = "sans",
  palette = NULL,
  shapes = NULL,
  linetypes = NULL,
  legend_position = "bottom"
)
```

Arguments

mode	"colour" or "grayscale".
variant	"publication" or compact "report".
base_size	Font size in points; values below 8 are rejected.
base_family	System-safe font family.
palette	Optional named qualitative palette.

shapes	Optional named shape mapping.
linetypes	Optional named line-type mapping.
legend_position	Legend position.

`cr_plot_timeline` *Time-course line plot*

Description

Time-course line plot

Usage

```
cr_plot_timeline(experiment, timepoint_var, channel, group_by = "treatment")
```

Arguments

experiment	A `cr_experiment`. Design table must have a time variable column.
timepoint_var	Name of the time variable column.
channel	Channel to plot.
group_by	Grouping variable (colour).

Value

A `ggplot` object.

See Also

Other visualisation: `cr_plot_comparison()`, `cr_plot_dose_response()`, `cr_plot_effect_sizes()`, `cr_plot_foldchange()`, `cr_plot_heatmap()`, `cr_plot_histogram()`, `cr_plot_intensity()`, `cr_plot_plate()`, `cr_plot_qc()`, `cr_plot_roc()`, `cr_plot_scatter()`, `cr_plot_spatial()`

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
exp$design$timepoint <- rep(c(0, 6, 12, 24), length.out = nrow(exp$design))
cr_plot_timeline(exp, "timepoint", "marker_1")
```

cr_power *Sample size for a future study, sized twice*

Description

Solves the two-sample design for the number of units per arm needed to reach power at sig_level, once for the observed effect and once for the confidence bound nearer the null.

Usage

```
cr_power(
  effect_size = NULL,
  ci_low = NULL,
  ci_high = NULL,
  power = 0.8,
  sig_level = 0.05,
  type = c("two.sample", "one.sample", "paired"),
  alternative = c("two.sided", "one.sided")
)
```

Arguments

effect_size	Numeric vector of observed standardized effects (Cohen's *d*). May be NULL when only interval bounds are given.
ci_low, ci_high	Numeric vectors of interval bounds for the same effects. Optional.
power	Target power (default 0.8).
sig_level	Significance level (default 0.05).
type	Design: "two.sample" (default), "one.sample" or "paired".
alternative	"two.sided" (default) or "one.sided".

Details

The conservative figure is the reportable one. Powering a screen's leading compound on its own point estimate is circular: that compound is the largest of the set only by virtue of having been selected for being largest, so a sample size derived from it can hardly fail to be met. n_conservative is NA whenever the interval spans the null, because an interval compatible with no effect cannot be sized.

The solver is stats::power.t.test(), so the effect size is read as a standardized mean difference and n is per group.

Value

A tibble with one row per effect: d_observed, n_observed, d_conservative, n_conservative, basis, power and sig_level. Sample sizes are units per arm, rounded up.

See Also

cr_power_grid(), cr_conservative_effect(), cr_power_analysis().

Other power: cr_conservative_effect(), cr_power_analysis(), cr_power_grid()

Examples

```
cr_power(effect_size = c(-1.31, -0.28),
         ci_low = c(-2.35, -1.20),
         ci_high = c(-0.27, 0.64))

# without an interval only the (circular) observed sizing is possible
cr_power(effect_size = 0.8)
```

---

cr_power_analysis *Simulated power under independent-cell assumptions*

Description

A Monte-Carlo approximation of the power of a two-sample comparison of replicate means formed from independent, unit-variance normal cells. There is no between-unit random effect or intraclass correlation. The result is an idealized scenario, not a validated estimate of power for a clustered biological assay or a formal upper bound on laboratory power. For a follow-up design use cr_power() with an effect on the independent experimental-unit scale, or a separately specified correlated-data model.

Usage

```
cr_power_analysis(
  effect_size,
  n_replicates,
  n_cells_per_rep,
  alpha = 0.05,
  test = "t_test",
  n_sim = 500,
  seed = NULL
)
```

Arguments

effect_size	Cohen's *d* at the cell level.
n_replicates	Number of replicate units per group.
n_cells_per_rep	Cells per replicate unit.
alpha	Type I error rate.
test	Only "t_test" is implemented.
n_sim	Number of simulations (default 500).
seed	Optional integer seed. The caller's random number stream is restored on exit.

Value

A tibble with the inputs and the simulated power.

See Also

cr_power().

Other power: cr_conservative_effect(), cr_power(), cr_power_grid()

Examples

```
cr_power_analysis(effect_size = 0.8, n_replicates = 4,
                  n_cells_per_rep = 100, n_sim = 100, seed = 1)
```

---

cr_power_grid *Sample sizes for a whole effect grid*

Description

Applies cr_power() to every row of an effect grid produced by cr_effect_grid() and binds the sizing columns onto it, so the observed and the conservative sample size travel with the contrast they belong to.

Usage

```
cr_power_grid(
  effects,
  estimate = "cohens_d",
  ci_low = NULL,
  ci_high = NULL,
  available = NULL,
  power = 0.8,
  sig_level = 0.05,
  type = c("two.sample", "one.sample", "paired"),
  alternative = c("two.sided", "one.sided")
)
```

Arguments

effects	An effect grid from cr_effect_grid(), or any data frame with an estimate column and its two interval columns.
estimate	Name of the effect size column. The interval columns default to <estimate>_ci_low and <estimate>_ci_high.
ci_low, ci_high	Optional explicit names of the interval columns.
available	Units already available per arm, used to flag which contrasts are already large enough. Either a column name in effects or a numeric vector. Defaults to pmin(n_ref, n_cmp) when those columns are present.

power	Target power (default 0.8).
sig_level	Significance level (default 0.05).
type	Design passed to cr_power().
alternative	Alternative passed to cr_power().

Value

effects with the columns d_conservative, n_observed, n_conservative and basis appended, plus n_available and sufficient when availability is known. Sufficiency requires a finite conservative target; no observed-effect fallback is used when the interval spans zero or has no usable bound. Numeric availability inputs must be scalar or have one value per effect.

See Also

cr_power(), cr_effect_grid().

Other power: cr_conservative_effect(), cr_power(), cr_power_analysis()

Examples

```
set.seed(1)
units <- data.frame(
  compound = rep(c("CompoundA", "CompoundB"), each = 12),
  arm = rep(rep(c("reference", "interval_short"), each = 6), 2),
  log2_fc = c(stats::rnorm(6, 0, 0.3), stats::rnorm(6, -1.4, 0.3),
              stats::rnorm(6, 0, 0.3), stats::rnorm(6, -0.1, 0.3))
)
eff <- cr_effect_grid(units, "log2_fc", "arm", "reference",
                      by = "compound")
sizes <- cr_power_grid(eff)
sizes[, c("compound", "n_observed", "n_conservative", "basis")]
```

---

cr_print_report_spec *Print a laboratory report specification*

Description

Print a laboratory report specification

Usage

```
cr_print_report_spec(x, ...)
```

Arguments

x	A cr_report_spec.
...	Unused.

Value

x, invisibly.

`cr_qc_doublets` *Flag or remove doublets*

Description

Flags cells whose size or DNA content is far above the population median, which in microscopy typically indicates segmented doublets. The cells are removed and recorded in the QC log.

Usage

```
cr_qc_doublets(
  experiment,
  channel = NULL,
  threshold_method = c("area", "channel"),
  k = 2.5
)
```

Arguments

`experiment`	A `cr_experiment`.
`channel`	Name of the channel column used when `threshold_method = "channel"` (for example a nuclear stain whose integrated intensity scales with DNA content). Ignored for `threshold_method = "area"`.
`threshold_method`	`"area"` (default) thresholds the segmentation area; `"channel"` thresholds `channel`.
`k`	Multiplicative threshold: cells whose value exceeds `k * median` are removed. Default `2.5`.

Value

A modified `cr_experiment`.

See Also

Other quality control: `cr_apply_gate()`, `cr_balance_cells()`, `cr_exclude_small()`, `cr_qc_filter()`, `cr_qc_gate()`, `cr_qc_gate_impact()`, `cr_qc_intensity()`, `cr_qc_manual()`, `cr_qc_report()`, `cr_qc_summary()`, `print.cr_qc_gate()`

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
cr_qc_doublets(exp, threshold_method = "area")
cr_qc_doublets(exp, channel = "DAPI", threshold_method = "channel")
```

`cr_qc_filter` *Filter cells by morphology*

Description

Removes debris and artifacts by thresholding morphological measurements. Any parameter left at `NA` is not applied.

Usage

```
cr_qc_filter(
  experiment,
  min_area = NA,
  max_area = NA,
  min_circularity = NA,
  max_circularity = NA
)
```

Arguments

`experiment`	A `cr_experiment`.
`min_area`, `max_area`	Numeric area thresholds. Cells outside the inclusive interval are removed.
`min_circularity`, `max_circularity`	Circularity thresholds (range 0-1).

Value

A modified `cr_experiment` with fewer cells and a QC log entry.

See Also

`cr_exclude_small()` for a data-derived (quantile) area threshold, and `cr_qc_summary()` for the log.

Other quality control: `cr_apply_gate()`, `cr_balance_cells()`, `cr_exclude_small()`, `cr_qc_doublets()`, `cr_qc_gate()`, `cr_qc_gate_impact()`, `cr_qc_intensity()`, `cr_qc_manual()`, `cr_qc_report()`, `cr_qc_summary()`, `print.cr_qc_gate()`

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 50)
exp2 <- cr_qc_filter(exp, min_area = 100, max_area = 2000)
cr_n_cells(exp2)
```

cr_qc_gate *Gate analysis units against their own in-batch control*

Description

A biological rather than a statistical gate: a treated unit must carry more target signal than the vehicle control acquired in its own batch. Units that do not are candidates for exclusion. The comparison is made against the control of the unit's *own* batch, so a plate that ran hot or cold as a whole cannot pass or fail units in unrelated batches.

Usage

```
cr_qc_gate(
  experiment,
  channel,
  control_level,
  batch_vars = NULL,
  unit = NULL,
  control_var = "treatment",
  statistic = c("median", "mean"),
  reference = c("median", "mean"),
  direction = c("greater", "less"),
  gate_controls = FALSE,
  min_cells = 1L
)
```

Arguments

experiment	A cr_experiment.
channel	Name of the raw signal column in cells.
control_level	Value of control_var marking the vehicle control units.
batch_vars	Character vector of columns (from cells or design) whose combination defines one batch. NULL (default) treats the whole experiment as a single batch.
unit	Column identifying the analysis unit. Defaults to the experiment's unit column, falling back to its spatial unit.
control_var	Column holding the treatment labels. Default "treatment".
statistic	Centre of the *unit* used for the comparison: "median" (default) or "mean".
reference	Centre of the *control* used as the threshold: "median" (default) or "mean".
direction	"greater" (default) requires a unit to exceed its control; "less" requires it to fall below.
gate_controls	Should control units be gated against themselves? Default FALSE; they form the reference arm.
min_cells	Units with fewer than min_cells cells fail regardless of their signal. Default 1, i.e. no effect.

Details

Two properties of this gate decide whether it is honest, and both are made explicit here.

Like for like. Target signal is usually right-skewed, so a control's mean sits above its median. Comparing a unit's median against the control's *mean* is therefore silently stricter than the rule it claims to apply. Because the gate can only ever drop *low* units, an over-strict gate manufactures apparent treatment effects. Both verdicts are computed for every unit — `fails_vs_median` and `fails_vs_mean` — and the units whose verdict *depends* on that choice are returned in `$disputed`. The default (`statistic = "median"`, `reference = "median"`) is like for like.

Leverage. Excluding a unit changes the estimate it fed into. Quantify that with `cr_qc_gate_impact()` *before* calling `cr_apply_gate()`, while both states are still in hand.

Gate the analysis unit, not the acquisition file: a unit assembled from two files would otherwise be gated twice.

Value

An object of class `cr_qc_gate`, a list with:

`units` One row per analysis unit: cell count, both raw centres, the control statistics of its batch, `pct_of_control`, `fails_vs_median`, `fails_vs_mean`, `disputed`, `verdict` and `reason`.

`disputed` The units whose verdict depends on which control centre is used.

`excluded` Identifiers of the units that fail under the chosen `statistic`/`reference` combination.

`params` The arguments the gate was built with.

See Also

`cr_qc_gate_impact()`, `cr_apply_gate()`, `cr_qc_report()`.

Other quality control: `cr_apply_gate()`, `cr_balance_cells()`, `cr_exclude_small()`, `cr_qc_doublets()`, `cr_qc_filter()`, `cr_qc_gate_impact()`, `cr_qc_intensity()`, `cr_qc_manual()`, `cr_qc_report()`, `cr_qc_summary()`, `print.cr_qc_gate()`

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 40)
gate <- cr_qc_gate(exp,
  channel = "marker_1",
  control_level = "Untreated",
  batch_vars = "replicate"
)
gate
head(gate$units[, c("well", "n_cells", "pct_of_control", "verdict")])
gate$excluded
```

`cr_qc_gate_impact` *Quantify the leverage of gate exclusions*

Description

Re-estimates every contrast affected by a gate exclusion twice — with the failing unit retained and with it removed — so that "one excluded unit changes the verdict" carries a number rather than a recollection. This matters most when an excluded unit sits in the *reference* arm, where retaining it makes the untreated condition look as though it were already responding.

Usage

```
cr_qc_gate_impact(
  experiment,
  gate,
  value = "log2_fc",
  group_var,
  reference_level,
  comparison_levels = NULL,
  by = NULL,
  unit = NULL,
  min_units = 3,
  affected_only = TRUE
)
```

Arguments

`experiment`	A `cr_experiment` that still contains the gated units.
`gate`	A `cr_qc_gate` from `cr_qc_gate()`.
`value`	Per-cell numeric column averaged to the unit level before the contrast is taken. Default `"log2_fc"`.
`group_var`	Column holding the arm labels of the contrast.
`reference_level`	Level of `group_var` used as the reference arm.
`comparison_levels`	Levels contrasted against the reference. `NULL` (default) uses every other level present.
`by`	Optional character vector of columns to compute the contrasts within (for example the compound).
`unit`	Analysis unit column. Defaults to the gate's unit.
`min_units`	Minimum number of units per arm; contrasts below it return `NA` estimates. Default `3`.
`affected_only`	Restrict the output to by groups that actually lost a unit. Default `TRUE`.

Details

Call this **before** cr_apply_gate(): the "retained" state needs the failing units to still be present in experiment.

Value

A tibble with one row per by group and contrast: unit counts in both states, estimate_with_excluded, estimate_without_excluded, their magnitudes, and whether the magnitude or the sign changes when the unit is dropped. The estimate is a pooled, uncorrected standardised mean difference.

See Also

cr_qc_gate(), cr_apply_gate().

Other quality control: cr_apply_gate(), cr_balance_cells(), cr_exclude_small(), cr_qc_doublets(), cr_qc_filter(), cr_qc_gate(), cr_qc_intensity(), cr_qc_manual(), cr_qc_report(), cr_qc_summary(), print.cr_qc_gate()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 40)
exp$cells$signal <- log2(exp$cells$marker_1)
gate <- cr_qc_gate(exp, "marker_1", "Untreated", batch_vars = "replicate")
cr_qc_gate_impact(exp, gate,
  value = "signal",
  group_var = "treatment",
  reference_level = "Untreated"
)
```

---

cr_qc_intensity *Gate cells by intensity*

Description

Removes cells whose intensity on a given channel is outside the interval [min_intensity, max_intensity].

Usage

```
cr_qc_intensity(experiment, channel, min_intensity = NA, max_intensity = NA)
```

Arguments

experiment	A cr_experiment.
channel	Name of the channel column in cells.
min_intensity, max_intensity	Lower and upper bounds. Leave NA to skip either.

Value

A modified `cr_experiment`.

See Also

`cr_qc_gate()` for a gate against each unit's own in-batch control rather than an absolute bound.

Other quality control: `cr_apply_gate()`, `cr_balance_cells()`, `cr_exclude_small()`, `cr_qc_doublets()`, `cr_qc_filter()`, `cr_qc_gate()`, `cr_qc_gate_impact()`, `cr_qc_manual()`, `cr_qc_report()`, `cr_qc_summary()`, `print.cr_qc_gate()`

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
cr_qc_intensity(exp, channel = "DAPI", min_intensity = 50)
```

---

cr_qc_manual *Manually exclude wells or cells*

Description

Removes entire wells/slides and/or specific cell IDs (for example contaminated wells or cells with focus issues).

Usage

```
cr_qc_manual(experiment, well = NULL, cell_ids = NULL)
```

Arguments

`experiment`	A `cr_experiment`.
`well`	Character vector of well / slide identifiers to remove. Default `NULL`.
`cell_ids`	Character vector of cell IDs to remove.

Value

A modified `cr_experiment`.

See Also

Other quality control: `cr_apply_gate()`, `cr_balance_cells()`, `cr_exclude_small()`, `cr_qc_doublets()`, `cr_qc_filter()`, `cr_qc_gate()`, `cr_qc_gate_impact()`, `cr_qc_intensity()`, `cr_qc_report()`, `cr_qc_summary()`, `print.cr_qc_gate()`

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 20)
cr_qc_manual(exp, well = c("A01", "H12"))
```

cr_qc_report *Report every analysis unit with its QC verdict*

Description

Returns one row per analysis unit so that the gate can be audited: which units were seen, what their signal was relative to their own control, which were excluded and why, and how many cells each still contributes. Units the gate removed are kept in the report — a gate whose casualties disappear from the record cannot be reviewed.

Usage

```
cr_qc_report(experiment, gate = NULL, unit = NULL, vars = NULL)
```

Arguments

experiment	A cr_experiment.
gate	Optional cr_qc_gate. Defaults to the gate stored by cr_apply_gate() in metadata$qc_gate, if any.
unit	Analysis unit column. Defaults to the gate's unit, or the experiment's unit / spatial unit.
vars	Optional character vector of extra design columns to carry into the report. NULL (default) carries the columns the gate was batched on.

Value

A tibble with one row per unit: the unit identifier, the requested design columns, n_cells_gated (cells the gate saw), n_cells (cells present now), retained, and — when a gate is available — the control statistics, pct_of_control, fails_vs_median, fails_vs_mean, disputed, verdict and reason. Failing units are listed first.

See Also

cr_qc_gate(), cr_apply_gate(), cr_qc_summary().

Other quality control: cr_apply_gate(), cr_balance_cells(), cr_exclude_small(), cr_qc_doublets(), cr_qc_filter(), cr_qc_gate(), cr_qc_gate_impact(), cr_qc_intensity(), cr_qc_manual(), cr_qc_summary(), print.cr_qc_gate()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 40)
gate <- cr_qc_gate(exp, "marker_1", "Untreated", batch_vars = "replicate")
exp2 <- cr_apply_gate(exp, gate)
rep <- cr_qc_report(exp2)
head(rep[, c("well", "n_cells", "verdict", "reason")])

# without a gate the report is still a per-unit inventory
head(cr_qc_report(exp))
```

cr_qc_summary *Summarise QC steps applied to an experiment*

Description

Summarise QC steps applied to an experiment

Usage

```
cr_qc_summary(experiment)
```

Arguments

experiment	A cr_experiment.

Value

A tibble with one row per QC step: step, parameters, cells_before, cells_after, cells_removed, percent_removed and timestamp.

See Also

cr_qc_report() for a per-unit rather than per-step view.

Other quality control: cr_apply_gate(), cr_balance_cells(), cr_exclude_small(), cr_qc_doublets(), cr_qc_filter(), cr_qc_gate(), cr_qc_gate_impact(), cr_qc_intensity(), cr_qc_manual(), cr_qc_report(), print.cr_qc_gate()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
exp <- cr_qc_filter(exp, min_area = 50)
cr_qc_summary(exp)
```

cr_read_cellprofiler *Read a CellProfiler object export*

Description

Imports a CellProfiler CSV export and renames the columns to the cellreportR convention (well, x, y, area, circularity, one column per marker channel).

Usage

```
cr_read_cellprofiler(path)
```

Arguments

path	Path to CellProfiler CSV.

Value

A tibble with standardised column names.

See Also

Other import: cr_assign_units(), cr_centroid_overlap(), cr_column_map(), cr_extract_markers(), cr_filename_grammar(), cr_marker_rules(), cr_merge_rules(), cr_parse_paths(), cr_path_spec(), cr_read_cells(), cr_read_design(), cr_read_export(), cr_read_exports(), cr_read_qupath(), cr_read_segmantr(), cr_unit_map()

Examples

```
f <- tempfile(fileext = ".csv")
utils::write.csv(
  data.frame(
    Metadata_Well = c("A01", "A01"),
    Location_Center_X = c(10, 20),
    Location_Center_Y = c(15, 25),
    AreaShape_Area = c(120, 130),
    Intensity_MeanIntensity_marker_1 = c(0.4, 0.6)
  ),
  f, row.names = FALSE
)
cr_read_cellprofiler(f)
```

---

cr_read_cells *Read segmented cell data from file*

Description

Reads per-cell measurements from CSV, TSV, Excel, RDS or FCS formats. The format is auto-detected from the file extension unless format is given explicitly.

Usage

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

Arguments

path	Path to file.
format	Optional format string: one of "csv", "tsv", "xlsx", "rds", "fcs". If NULL, inferred from file extension.

Value

A tibble of cell measurements.

See Also

`cr_read_export()` and `cr_read_exports()` for exports that carry design information in their path and file name.

Other import: `cr_assign_units()`, `cr_centroid_overlap()`, `cr_column_map()`, `cr_extract_markers()`, `cr_filename_grammar()`, `cr_marker_rules()`, `cr_merge_rules()`, `cr_parse_paths()`, `cr_path_spec()`, `cr_read_cellprofiler()`, `cr_read_design()`, `cr_read_export()`, `cr_read_exports()`, `cr_read_qupath()`, `cr_read_segmantr()`, `cr_unit_map()`

Examples

```
f <- tempfile(fileext = ".csv")
utils::write.csv(
  data.frame(cell_id = "c1", well = "A01", target_signal = 12),
  f, row.names = FALSE
)
cr_read_cells(f)
```

---

cr_read_design *Read experimental design from CSV or Excel*

Description

Read experimental design from CSV or Excel

Usage

```
cr_read_design(path)
```

Arguments

`path`	Path to file. `.csv`, `.tsv` and `.xlsx` are supported.

Value

A tibble of design information.

See Also

`cr_design()` to turn the table into a validated design object.

Other import: `cr_assign_units()`, `cr_centroid_overlap()`, `cr_column_map()`, `cr_extract_markers()`, `cr_filename_grammar()`, `cr_marker_rules()`, `cr_merge_rules()`, `cr_parse_paths()`, `cr_path_spec()`, `cr_read_cellprofiler()`, `cr_read_cells()`, `cr_read_export()`, `cr_read_exports()`, `cr_read_qupath()`, `cr_read_segmantr()`, `cr_unit_map()`

Examples

```
f <- tempfile(fileext = ".csv")
utils::write.csv(
  data.frame(well = c("A01", "A02"), treatment = c("Vehicle", "CompoundA")),
  f, row.names = FALSE
)
cr_read_design(f)
```

`cr_read_export` *Read one segmented single-cell export*

Description

Reads a single export file (`.csv`, `.tsv`, `.txt`, `.xls` or `.xlsx`), applies an optional column contract and prepends the file's provenance.

Usage

```
cr_read_export(
  path,
  column_map = NULL,
  drop_empty_rows = TRUE,
  col_types = NULL,
  sheet = 1L,
  call = rlang::caller_env()
)
```

Arguments

`path`	Path to the export file.
`column_map`	Optional `cr_column_map()` describing how raw headers translate to analysis names.
`drop_empty_rows`	Logical. Drop rows that are `NA` in every measurement column. Default `TRUE`.
`col_types`	Optional column-type specification passed to the underlying reader (a `readr` column specification for delimited files, a `readxl` type string such as `"numeric"` for Excel files).
`sheet`	Sheet name or index for Excel exports. Default `1`.
`call`	The execution environment of the calling function. Used for error reporting; experts only.

Details

Provenance is not optional in this reader. One export is one acquisition of one spatial unit, and export base names repeat across plates, so the *path* — not the file name — identifies the acquisition. Both are carried on every row as `source_file` and `source_path`, and both survive into the analysis unit assignment performed by `cr_assign_units()`.

Many instruments terminate an export with a single all-blank row. With `drop_empty_rows = TRUE` (the default) such rows are removed rather than carried into the analysis as an all-`NA` cell.

Value

A tibble of cells with `source_file` and `source_path` as the first two columns.

See Also

`cr_read_exports()` for a whole directory tree, `cr_column_map()`.

Other import: `cr_assign_units()`, `cr_centroid_overlap()`, `cr_column_map()`, `cr_extract_markers()`, `cr_filename_grammar()`, `cr_marker_rules()`, `cr_merge_rules()`, `cr_parse_paths()`, `cr_path_spec()`, `cr_read_cellprofiler()`, `cr_read_cells()`, `cr_read_design()`, `cr_read_exports()`, `cr_read_qupath()`, `cr_read_segmantr()`, `cr_unit_map()`

Examples

```
d <- file.path(tempdir(), "cr_export_demo")
dir.create(d, showWarnings = FALSE)
raw <- data.frame(
  "Event Label" = 1:3,
  "Signal - Mean Intensity" = c(120, 140, 95),
  check.names = FALSE
)
raw[4, ] <- NA # trailing blank row, as many instruments write
f <- file.path(d, "CompoundA_5min_10uM_treated_1.csv")
utils::write.csv(raw, f, row.names = FALSE)

map <- cr_column_map(
  exact = c("Event Label" = "cell_id",
            "Signal - Mean Intensity" = "target_signal")
)
cr_read_export(f, column_map = map)
```

---

cr_read_exports *Read a directory tree of segmented single-cell exports*

Description

Walks a directory recursively, reads every file matching `pattern` with `cr_read_export()` and row-binds the result. Design facts that live in the directory layout and in the file names — rather than inside the files — are recovered by a parser and joined back onto the cells.

Usage

```
cr_read_exports(
  root,
  pattern = "\\.(csv|tsv|xls|xlsx)$",
  column_map = NULL,
  spec = NULL,
  parser = NULL,
  recursive = TRUE,
  drop_empty_rows = TRUE,
  col_types = NULL,
  progress = TRUE,
  call = rlang::caller_env()
)
```

Arguments

root	Directory to walk.
pattern	Regular expression selecting export files. Default matches .csv, .tsv, .xls and .xlsx.
column_map	Optional cr_column_map() applied to every export.
spec	Optional cr_path_spec() describing the directory and file-name grammar. Ignored when parser is supplied.
parser	Optional function of the file paths returning a tibble with one row per path.
recursive	Logical. Walk sub-directories. Default TRUE.
drop_empty_rows	Logical. Passed to cr_read_export().
col_types	Optional column-type specification passed to cr_read_export().
progress	Logical. Show a progress bar while reading. Default TRUE.
call	The execution environment of the calling function. Used for error reporting; experts only.

Details

No naming convention is hardcoded. Either supply a cr_path_spec() via spec, which is handed to cr_parse_paths(), or supply an arbitrary parser function. A parser receives the character vector of file paths and must return one row per path; if it returns a source_path column the join is made on that column, otherwise row order is assumed to match.

Row-binding is tolerant of exports that carry different subsets of the available columns: absent columns are filled with NA.

Value

A tibble of all cells, with source_file and source_path provenance columns and any columns produced by the parser. The paths that were read are attached as the "files" attribute.

See Also

cr_read_export(), cr_path_spec(), cr_parse_paths(), cr_dataset().

Other import: cr_assign_units(), cr_centroid_overlap(), cr_column_map(), cr_extract_markers(), cr_filename_grammar(), cr_marker_rules(), cr_merge_rules(), cr_parse_paths(), cr_path_spec(), cr_read_cellprofiler(), cr_read_cells(), cr_read_design(), cr_read_export(), cr_read_qupath(), cr_read_segmantr(), cr_unit_map()

Examples

```
# Build a small two-file export tree.
root <- file.path(tempdir(), "cr_exports_demo")
unlink(root, recursive = TRUE)
leaf <- file.path(root, "Run1", "CompoundA", "Plate_1")
dir.create(leaf, recursive = TRUE, showWarnings = FALSE)
one <- function(n) {
  data.frame("Event Label" = seq_len(n),
             "Signal - Mean Intensity" = seq_len(n) * 10,
             check.names = FALSE)
}
utils::write.csv(one(3), file.path(leaf, "CompoundA_vehicle_1.csv"),
                 row.names = FALSE)
utils::write.csv(one(4), file.path(leaf, "CompoundA_5min_10uM_treated_1.csv"),
                 row.names = FALSE)

spec <- cr_path_spec(
  levels = c("run", "compound", "plate"),
  grammar = cr_filename_grammar(
    tokens = list(interval = "[0-9]+min", dose = "[0-9]+uM"),
    defaults = list(interval = "none", dose = "vehicle"),
    prefix_strip = "CompoundA"
  )
)
cells <- cr_read_exports(root, spec = spec, progress = FALSE)
cells
```

---

cr_read_qupath *Read a QuPath measurement export*

Description

Read a QuPath measurement export

Usage

```
cr_read_qupath(path)
```

Arguments

path	Path to QuPath TSV.

Value

A tibble with standardised column names.

See Also

Other import: cr_assign_units(), cr_centroid_overlap(), cr_column_map(), cr_extract_markers(), cr_filename_grammar(), cr_marker_rules(), cr_merge_rules(), cr_parse_paths(), cr_path_spec(), cr_read_cellprofiler(), cr_read_cells(), cr_read_design(), cr_read_export(), cr_read_exports(), cr_read_segmantr(), cr_unit_map()

Examples

```
f <- tempfile(fileext = ".tsv")
writeLines(
  c("Image\tCentroid X um\tCentroid Y um\tArea um^2\tCell: marker_1 mean",
    "A01.ome.tif\t10\t12\t120\t0.44",
    "A01.ome.tif\t30\t42\t118\t0.51"),
  f
)
cr_read_qupath(f)
```

---

cr_read_segmantr *Read a segmantR result*

Description

Imports an RDS file produced by the segmantR segmentation package. Accepts either a segmantr_result list (where cells are in $cells) or a plain data frame.

Usage

```
cr_read_segmantr(path)
```

Arguments

path	Path to RDS file.

Value

A tibble of cells.

See Also

Other import: cr_assign_units(), cr_centroid_overlap(), cr_column_map(), cr_extract_markers(), cr_filename_grammar(), cr_marker_rules(), cr_merge_rules(), cr_parse_paths(), cr_path_spec(), cr_read_cellprofiler(), cr_read_cells(), cr_read_design(), cr_read_export(), cr_read_exports(), cr_read_qupath(), cr_unit_map()

Examples

```
f <- tempfile(fileext = ".rds")
saveRDS(data.frame(cell_id = "c1", well = "A01", target_signal = 12), f)
cr_read_segmantr(f)
```

`cr_render_lab_report` *Render a concise laboratory report*

Description

Render a concise laboratory report

Usage

```
cr_render_lab_report(
  report,
  output_file,
  template = NULL,
  quiet = TRUE,
  audit_file = NULL,
  overwrite = FALSE,
  keep_tex = FALSE
)
```

Arguments

`report`	A `cr_lab_report`.
`output_file`	Explicit output file ending in `.pdf` or `.html`.
`template`	Optional R Markdown template. The default HTML template embeds its assets, disables dynamic math rendering, and wraps long table values. Custom templates retain their own HTML output options.
`quiet`	Passed to `rmarkdown::render()`.
`audit_file`	Optional JSON audit path written after rendering. It must be distinct from the report path, including relative-path aliases.
`overwrite`	Replace existing report and audit files. Defaults to `FALSE`, so released or draft reports are never silently replaced.
`keep_tex`	Keep the generated `.tex` file beside a PDF for layout diagnostics. Defaults to `FALSE`.

`cr_render_report` *Render a report to HTML or PDF*

Description

Renders an assembled report through an R Markdown template. The bundled template covers experimental setup, the QC log, an intensity overview, the comparison summary and a session-info appendix. A custom template receives the report through its `params`; only the parameters a template declares are passed, so templates of different vintages keep working.

Usage

```
cr_render_report(
  report,
  output_dir = tempdir(),
  format = c("html", "pdf", "docx"),
  template = NULL,
  title = NULL,
  author = NULL,
  quiet = TRUE
)
```

Arguments

`report`	A `cr_report` from `cr_report()`. A `cr_experiment` is also accepted and is assembled into a report first.
`output_dir`	Output directory (default `tempdir()`).
`format`	One of `"html"`, `"pdf"` or `"docx"`.
`template`	Path to an R Markdown template. If `NULL`, the bundled template is used.
`title, author`	Override the report title and author. `NULL` keeps the values stored in the report.
`quiet`	Passed to `rmarkdown::render()`.

Value

Path to the rendered document (invisibly).

See Also

`cr_report()`.

Other reporting: `cr_export_plots()`, `cr_export_results()`, `cr_export_tables()`, `cr_report()`, `cr_table_qc()`, `cr_tables()`

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 10)
rep <- cr_report(exp)
if (requireNamespace("rmarkdown", quietly = TRUE) &&
    requireNamespace("knitr", quietly = TRUE) &&
    rmarkdown::pandoc_available()) {
  out <- cr_render_report(rep, output_dir = tempdir())
  basename(out)
}
```

`cr_report` *Assemble a structured analysis report*

Description

Collects everything a write-up needs into one `cr_report` object: the experiment, the statistical results, the quality-control record, the effect-size grid, the sample-size table, any supplementary tables and any plots. Assembling first and formatting later is what keeps a number in the text tied to the object it came from — tables, generated macros and the rendered document are then all derived from the same assembly rather than transcribed from each other.

Usage

```
cr_report(
  experiment,
  results = NULL,
  qc = NULL,
  effects = NULL,
  sizes = NULL,
  tables = NULL,
  plots = NULL,
  title = "cellreportR analysis report",
  author = "",
  metadata = list(),
  render = NULL,
  template = NULL,
  output_dir = NULL,
  format = c("html", "pdf", "docx")
)
```

Arguments

`experiment`	A `cr_experiment`.
`results`	Optional `cr_result`, list of `cr_result` objects (as returned by `cr_test_all()`), or a data frame of results.

qc	Optional quality-control record: a data frame, or a list holding one (for example a gate object with a units element). If NULL, the experiment's QC log is used.
effects	Optional data frame of effect sizes — one row per contrast, typically with estimate and confidence-bound columns.
sizes	Optional data frame of sample-size calculations, one row per contrast.
tables	Optional named list of supplementary tables (data frames). A disposition table is added automatically when the list does not already contain one.
plots	Optional named list of ggplot2 objects.
title	Report title.
author	Author name for the report header.
metadata	Optional named list of arbitrary metadata.
render	Whether to render the report to a file. The default NULL renders when template or output_dir is supplied — that is, when the caller has said where the document should go — and otherwise returns the assembled object. Pass TRUE or FALSE to be explicit.
template	Path to an R Markdown template. If NULL, the bundled template is used.
output_dir	Output directory for the rendered document.
format	One of "html", "pdf" or "docx".

Value

A cr_report object, an S3 list with the slots

experiment The cr_experiment the report describes.

results Named list of cr_result objects.

summary One-row-per-contrast overview tibble.

qc Quality-control tibble.

effects Effect-size tibble, or NULL.

sizes Sample-size tibble, or NULL.

tables Named list of supplementary tibbles.

plots Named list of ggplot2 objects.

metadata User metadata.

params Title, author, package version, creation time.

When rendering was requested, the path to the rendered document is returned invisibly instead, carrying the assembled object in its report attribute.

See Also

cr_render_report() to render an assembled report, cr_tables() to extract its tables and cr_macros() to emit its numbers.

Other reporting: cr_export_plots(), cr_export_results(), cr_export_tables(), cr_render_report(), cr_table_qc(), cr_tables()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 20)
rep <- cr_report(exp, title = "Marker 1 overview")
rep

# supply an effect grid and the sample sizes derived from it
eff <- data.frame(
  group = c("CompoundA_low", "CompoundA_high"),
  estimate = c(0.31, 1.42),
  ci_low = c(-0.10, 0.55),
  ci_high = c(0.72, 2.29)
)
rep2 <- cr_report(exp, effects = eff, title = "Marker 1 screen")
rep2$summary
```

---

cr_report_data *Inspect the exact structured values entering a laboratory report*

Description

Inspect the exact structured values entering a laboratory report

Usage

```
cr_report_data(report)
```

Arguments

report	A cr_lab_report.

---

cr_report_display_data *Prepare human-facing values for report rendering*

Description

Produces already formatted labels, dates, values, section visibility flags, and audit rows. It does not change the report specification.

Usage

```
cr_report_display_data(report)
```

Arguments

report	A cr_lab_report.

Value

A plain nested list suitable for inspection, serialization, and deterministic rendering.

See Also

Other laboratory reporting: `cr_report_profile()`, `cr_report_spec()`, `cr_report_style()`, `cr_validate_report_spec()`

`cr_report_evidence` *Bind report claims and figures to portable source-file evidence*

Description

This neutral provenance layer does not calculate effects, infer units, verify scientific validity, or prove that a figure was generated from its declared tables. Figure links are explicit declarations whose file identities are checked. Source files remain external to the report.

Usage

```
cr_report_evidence(root, tables, claims = list(), figures = list())
```

Arguments

`root`	Existing directory containing all source files. Paths stored in the evidence are relative to this root, so the complete tree can be moved.
`tables`	Named list of descriptors with `path`, `format` (`"csv"` or `"tsv"`), and `key` (one or more unique-key column names). Fields are read as UTF-8 character strings without automatic missing-value conversion.
`claims`	Named list of descriptors with `table`, `key` (a named character vector giving every key value), `column`, and optional `format` (`"text"` or `"number"`, default `"text"`) and `digits` (default 3). Numeric claims must be finite decimal numbers; formatting uses `cr_format_number()`. No scaling, unit conversion, missing-value substitution or inference occurs.
`figures`	Named list of descriptors with `path` and `tables` (nonempty character vector of declared source-table identifiers).

Value

`cr_report_evidence` returns a versioned list with file SHA-256 hashes, keyed raw values, formatted displays and a manifest SHA-256. Its hash uses UTF-8 JSON bytes, not R serialization. Identifiers use radix ordering and numeric displays use a period decimal mark regardless of locale/options. Row order and other source-byte changes deliberately invalidate the source hash.

Examples

```
root <- tempfile(); dir.create(root)
writeLines(c("group,value", "control,1.25"), file.path(root, "result.csv"))
evidence <- cr_report_evidence(root,
  tables = list(result = list(path = "result.csv", format = "csv", key = "group")),
  claims = list(mean = list(table = "result", key = c(group = "control"),
                            column = "value", format = "number", digits = 2)))
cr_validate_report_evidence(evidence, root)
unlink(root, recursive = TRUE)
```

---

cr_report_hash *Hash report-relevant structured data*

Description

Hash report-relevant structured data

Usage

```
cr_report_hash(x, algorithm = "sha256")
```

Arguments

x	Any R object.
algorithm	Algorithm supported by digest::digest(), default SHA-256.

---

cr_report_profile *Create a reusable laboratory report profile*

Description

A profile stores organization-wide branding and display policy separately from examination-specific scientific configuration.

Usage

```
cr_report_profile(
  laboratory = list(),
  style = cr_report_style(),
  labels = character(),
  footer_statement = NULL,
  default_title = NULL,
  required_fields = character()
)
```

Arguments

laboratory	Default laboratory metadata.
style	A cr_report_style().
labels	Named display-label overrides.
footer_statement	Optional laboratory-controlled release statement.
default_title	Optional default report title.
required_fields	Optional dotted required-field paths.

Value

A cr_report_profile object.

See Also

Other laboratory reporting: cr_report_display_data(), cr_report_spec(), cr_report_style(), cr_validate_report_spec()

---

cr_report_provenance *Build machine-readable report provenance*

Description

Build machine-readable report provenance

Usage

```
cr_report_provenance(
  experiment = NULL,
  report_spec,
  output_file = NULL,
  analysis_metadata = list()
)
```

Arguments

experiment	Optional cr_experiment.
report_spec	A cr_report_spec or cr_lab_report.
output_file	Optional rendered file whose byte hash is recorded.
analysis_metadata	Optional concise metadata about upstream result objects.

cr_report_qc *Construct a concise report QC table*

Description

Construct a concise report QC table

Usage

```
cr_report_qc(
  criterion = character(),
  observed = character(),
  acceptance = character(),
  status = character(),
  source = character()
)
```

Arguments

criterion, observed, acceptance, status	Vectors defining reportable QC rows.
source	Optional source identifiers.

cr_report_qc_from_log *Convert an existing experiment QC log to report-compatible rows*

Description

Convert an existing experiment QC log to report-compatible rows

Usage

```
cr_report_qc_from_log(x)
```

Arguments

x	A cr_experiment or QC-log data frame.

`cr_report_review_app` *Review an immutable source-bound report in a Shiny application*

Description

Creates an application object without starting a listener. The report is detached from the caller by serialization; previews and downloads validate its captured state and bound source files before and after each operation. Claim values, source metadata, limitations and report status are visible. Source details expand using keyboard-accessible disclosures on narrow screens.

Usage

```
cr_report_review_app(report, root = NULL)
```

Arguments

`report`	An already-bound `cr_lab_report` with `cr_report_evidence`. Experiment payloads, live result graphics, external logos and mutable objects such as environments, functions or connections are not supported. Trusted HTML, tags and other unsupported S3 classes are rejected, including in attributes. Supported classes are the package's report, evidence, spec, QC, style and profile classes, lists, data frames and tibbles, Date, POSIXt and factors. Unclassed atomic vectors and lists are also supported.
`root`	Optional existing source root for identical relocated evidence. Defaults to the report's bound evidence root.

Details

The optional package `bslib` is required. HTML/PDF downloads additionally require the render dependencies of `cr_render_lab_report()`, including `rmarkdown`, Pandoc and a suitable LaTeX installation for PDF.

Only declared PNG/PDF figures are downloadable. Filename extensions and byte signatures must agree; this is not full decoding or content sanitization. Source tables are shown as metadata and are never registered as a static directory. Staged downloads are removed after failure or completion.

This application is not a sandbox, authentication service or authorization system. Source identity does not establish scientific validity or release authorization. Pre/post checks do not provide filesystem snapshot isolation. The function starts no server and changes no global listener configuration. For local review, explicitly use `shiny::runApp(app, host = "127.0.0.1")`. Claims and figures may be empty; bound evidence requires source tables.

Value

A `shiny.appobj`, suitable for `shiny::runApp()`.

See Also

`cr_report_evidence()`, `cr_lab_report_from_evidence()`

Examples

```
if (interactive() && requireNamespace("bslib", quietly = TRUE)) {
  root <- tempfile("report-sources-")
  dir.create(root)
  writeLines(c("group,value", "treated,1.25"), file.path(root, "results.csv"))
  evidence <- cr_report_evidence(root,
    tables = list(results = list(
      path = "results.csv", format = "csv",
      key = "group"
    )),
    claims = list(estimate = list(
      table = "results",
      key = c(group = "treated"), column = "value"
    ))
  )
  spec <- cr_report_spec(
    report = list(report_id = "example"),
    result = list(display_value = "Source-bound example")
  )
  report <- cr_lab_report_from_evidence(evidence, root, spec)
  app <- cr_report_review_app(report)
  # shiny::runApp(app, host = "127.0.0.1")
}
```

---

cr_report_spec *Construct a configurable laboratory report specification*

Description

Creates a domain-neutral, versioned description of values that may appear in a laboratory-style report. No analytical calculation or interpretation is performed. All scientific content is supplied by the caller.

Usage

```
cr_report_spec(
  report = list(),
  laboratory = list(),
  subject = list(),
  specimen = list(),
  examination = list(),
  result = list(),
  interpretation = list(),
  limitations = character(),
  authorization = list(),
  custom_fields = list(),
  custom_sections = list(),
  required_fields = character(),
```

```
  field_labels = character(),
  schema_version = "1.0"
)
```

Arguments

report, laboratory, subject, specimen, examination, result, interpretation, authorization	Named lists for the corresponding report sections.
limitations	Character vector of user-supplied limitations.
custom_fields	Named list of additional label/value pairs.
custom_sections	List of sections, each with `title` and named `fields`.
required_fields	Character vector of dotted field paths required by the caller.
field_labels	Named character vector of optional display-label overrides.
schema_version	Report-data schema version, independent of the package version.

Value

An object of class `cr_report_spec`.

See Also

Other laboratory reporting: `cr_report_display_data()`, `cr_report_profile()`, `cr_report_style()`, `cr_validate_report_spec()`

---

cr_report_style *Configure the appearance of laboratory reports*

Description

Creates a presentation-only object shared by programmatic rendering and the Shiny report workflow. It does not alter report values or analytical data.

Usage

```
cr_report_style(
  paper = "A4",
  mode = c("colour", "grayscale"),
  density = c("standard", "compact"),
  locale = c("en", "de"),
  logo = NULL,
  primary_colour = "#315A70",
  secondary_colour = "#65747C",
  date_format = NULL,
```

```
  date_time_format = NULL,
  labels = character(),
  footer_text = NULL,
  include_audit_appendix = FALSE,
  show_signature_lines = FALSE,
  draft_watermark = FALSE
)
```

Arguments

`paper`	Paper size. Version 2 currently supports `"A4"`.
`mode`	`"colour"` or intentionally monochrome `"grayscale"`.
`density`	`"standard"` or `"compact"`.
`locale`	Display-label locale, currently `"en"` or `"de"`.
`logo`	Optional local PNG, JPEG, or PDF logo path.
`primary_colour, secondary_colour`	Six-digit hex colours. In grayscale mode neutral values are used regardless of these settings.
`date_format, date_time_format`	Optional `format()` patterns. Defaults are human-readable and locale-specific.
`labels`	Named display-label overrides.
`footer_text`	Optional laboratory-controlled document-footer statement.
`include_audit_appendix`	Include the technical appendix by default.
`show_signature_lines`	Add printable reviewer/authorizer lines.
`draft_watermark`	Add a light watermark only when status is `DRAFT`.

Value

A `cr_report_style` object.

See Also

Other laboratory reporting: `cr_report_display_data()`, `cr_report_profile()`, `cr_report_spec()`, `cr_validate_report_spec()`

cr_roc — *Extract or compute an ROC curve from a* `cr_result`

Description

Extract or compute an ROC curve from a `cr_result`

Usage

```
cr_roc(result)
```

Arguments

`result`	A `cr_result` produced by `cr_logistic()`.

Value

A tibble with `threshold`, `sensitivity`, `specificity`, `fpr` and `tpr`.

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
res <- cr_logistic(exp, "marker_1", "CompoundA_high", "Untreated")
head(cr_roc(res))
```

cr_rowcol_to_well — *Convert row and column indices to well IDs*

Description

Convert row and column indices to well IDs

Usage

```
cr_rowcol_to_well(row, col, pad = 2)
```

Arguments

`row`	Integer vector of row numbers (1 = A, 2 = B, ...).
`col`	Integer vector of column numbers.
`pad`	Number of digits to pad the column number to (default 2, yielding `"A01"` rather than `"A1"`).

Value

A character vector of well IDs.

Examples

```
cr_rowcol_to_well(c(1, 2, 8), c(1, 5, 12))
```

`cr_run_app` *Launch the cellreportR Shiny application*

Description

Starts the interactive front-end bundled with the package. The app covers the whole analysis path in one place: point it at a directory of segmented exports (or upload the files), specify or edit the design, review quality control with a per-unit exclusion table, choose a normalisation, estimate effect sizes as a table plus forest plot, and export a report.

Usage

```
cr_run_app(
  experiment = NULL,
  report_spec = NULL,
  report_profile = NULL,
  max_upload_mb = 512,
  launch_browser = interactive(),
  ...
)
```

Arguments

`experiment`	Optional `cr_experiment` to load on start-up. It is validated with `cr_validate_experiment()` before the app is built and handed to the server through the app object, so no global state is involved. When `NULL` (default) the app starts empty and offers synthetic example data.
`report_spec`	Optional `cr_report_spec()` used to pre-populate the laboratory-report workflow.
`report_profile`	Optional `cr_report_profile()` providing reusable laboratory metadata, labels, required fields, and visual style.
`max_upload_mb`	Numeric. Maximum upload size per request, in megabytes. Segmented single-cell exports routinely exceed the Shiny default of 5 MB; this argument raises the limit for the duration of the running app and restores the previous value on exit. Default `512` MB.
`launch_browser`	Whether to open a browser window. Defaults to `TRUE` in interactive sessions.
`...`	Further arguments passed to `shiny::runApp()`.

Details

The user interface, the server and every helper live inside the package namespace, so the app can be driven in-process by `shiny::testServer()` as well as from a browser. The directory under `inst/shiny/cellreportR/` holds the `www/` assets and a thin `app.R` shim for deployment on a Shiny server.

Requires the `shiny`, `bslib`, `DT` and `ggplot2` packages, all listed under `Suggests`. An informative error naming the missing package is raised when any of them is unavailable.

Value

Invoked for its side effect of running the app. Returns whatever `shiny::runApp()` returns, invisibly.

See Also

`cr_build_experiment()`, `cr_report()`

Examples

```
if (interactive()) {
  cr_run_app()
}

# Start from an experiment that is already in the session:
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 10)
if (interactive()) {
  cr_run_app(exp)
}
```

---

cr_save_figure *Save a publication-ready figure deterministically*

Description

Save a publication-ready figure deterministically

Usage

```
cr_save_figure(
  plot,
  filename,
  size = c("single", "double", "report", "square"),
  dpi = 600,
  background = "white",
  metadata = NULL
)
```

Arguments

plot	A ggplot object.
filename	Explicit PDF, SVG, PNG, TIFF or TIF filename.
size	Standard size name or numeric width/height.
dpi	Raster resolution.
background	Explicit background.
metadata	Optional list written to a JSON sidecar.

Details

TIFF uses Cairo with lossless LZW compression when native Cairo and TIFF support are available. Otherwise the normal bitmap device is used. With Quartz configured, the fallback is uncompressed because Quartz does not support compression. CRAN R on macOS needs XQuartz for Cairo devices. A reported Cairo capability alone does not establish that the runtime libraries are installed. The figure and optional sidecar are staged before replacing their targets. Missing or empty device output is an error. Ordinary publication failures restore previous files; callers must serialize writers to these paths.

---

cr_save_plot *Save a figure at publication settings*

Description

Writes one plot to one or more formats with the package's publication defaults: 600 dpi, a white background (so a transparent panel does not render as black in a typeset document) and a golden-ratio height when none is given.

Usage

```
cr_save_plot(
  plot,
  path,
  width = 12,
  height = NULL,
  dpi = 600,
  formats = c("png", "pdf"),
  units = "in",
  bg = "white",
  quiet = FALSE
)
```

Arguments

plot	A ggplot object, or any object ggplot2::ggsave() accepts.
path	Output path. Any extension is replaced by each entry of formats, so one call can write matching raster and vector files.
width	Width in units (default 12).
height	Height in units. NULL (default) uses width / 1.618.
dpi	Raster resolution (default 600).
formats	Character vector of file extensions to write (default c("png", "pdf")).
units	Passed to ggplot2::ggsave() (default "in").
bg	Background colour (default "white").
quiet	Suppress the message naming the written files (default FALSE).

Value

The written paths, invisibly, as a character vector.

See Also

Other screen figures: cr_plot_forest(), cr_plot_qc_gate(), cr_plot_sample_size(), cr_plot_screen(), cr_plot_specificity()

Examples

```
p <- ggplot2::ggplot(data.frame(x = 1:5, y = 1:5),
                     ggplot2::aes(x, y)) +
  ggplot2::geom_point() +
  cr_theme()
out <- file.path(tempdir(), "figure-1")
paths <- cr_save_plot(p, out, width = 5, formats = "png")
basename(paths)
unlink(paths)
```

---

cr_scale_colour *Manual colour scale from the cellreportR visual system*

Description

Manual colour scale from the cellreportR visual system

Usage

```
cr_scale_colour(values = NULL, mode = c("colour", "grayscale"), ...)

cr_scale_fill(values = NULL, mode = c("colour", "grayscale"), ...)

cr_shape_scale(values = NULL, ...)

cr_linetype_scale(values = NULL, ...)
```

Arguments

values	Optional named values.
mode	Colour mode.
...	Passed to ggplot2::scale_colour_manual().

---

cr_scale_diverging *Continuous diverging scale*

Description

Continuous diverging scale

Usage

```
cr_scale_diverging(
  midpoint = 0,
  aesthetics = c("colour", "fill"),
  mode = c("colour", "grayscale"),
  ...
)
```

Arguments

midpoint	Explicit neutral midpoint.
aesthetics	"colour" or "fill".
mode	Colour mode.
...	Passed to the gradient scale.

---

cr_scale_group *Grouping scales with redundant encoding*

Description

Builds the discrete scales that carry a grouping variable: a colour-vision-safe hue from cr_palette() plus a matching symbol from cr_shapes(). Add the result to a plot with +, exactly like a single scale.

Usage

```
cr_scale_group(
  aesthetics = c("colour", "fill", "shape"),
  name = NULL,
  guide_for = NULL,
  drop = TRUE,
  mode = c("colour", "grayscale")
)
```

Arguments

aesthetics	Character vector of aesthetics to scale. Any of "colour", "color", "fill", "shape".
name	Legend title. NULL (default) keeps the mapped variable name.
guide_for	Aesthetics that keep a visible guide. Use "all" when the aesthetics are drawn by the same layer and should merge into one key, "none" to suppress every guide. Defaults to the last of aesthetics, which is the one drawn on top.
drop	Passed to the underlying scales: drop unused factor levels (default TRUE).
mode	Colour or grayscale palette for colour/fill aesthetics.

Details

Because the same variable is mapped to several aesthetics, the guide is emitted for one aesthetic only. A duplicate guide is what makes a composed figure print the same legend twice, so guide_for names the single aesthetic that keeps its legend and every other aesthetic is suppressed.

Value

A list of ggplot2 scales.

See Also

cr_palette(), cr_shapes(), cr_theme()

Other plot design: cr_palette(), cr_shapes(), cr_theme()

Examples

```
df <- data.frame(g = rep(c("a", "b", "c"), each = 10), y = rnorm(30))
ggplot2::ggplot(df, ggplot2::aes(g, y, colour = g, shape = g)) +
  ggplot2::geom_point() +
  cr_scale_group(c("colour", "shape"), name = "group") +
  cr_theme()
```

---

cr_shapes *Redundant shape encoding*

Description

Returns plotting symbols matched to cr_palette(), so that a grouping encoded by hue is also encoded by shape. Filled shapes (21:25, which take both a colour and a fill) come first, then open shapes.

Usage

```
cr_shapes(n = NULL, names = NULL)
```

Arguments

n	Number of shapes to return. NULL (default) returns all 10. Requesting more recycles and warns, because shape has then stopped separating the levels.
names	Optional character vector of names to attach, for use as the values argument of ggplot2::scale_shape_manual().

Value

An integer vector of ggplot2 shape codes.

See Also

cr_palette(), cr_scale_group()

Other plot design: cr_palette(), cr_scale_group(), cr_theme()

Examples

```
cr_shapes(4)
cr_shapes(3, names = c("low", "mid", "high"))
```

---

cr_standardize_batch *Standardize a channel against the control of each cell's own batch*

Description

Expresses every cell relative to the control cells of the batch it belongs to, rather than to one pooled reference for the whole experiment. This is the standardization a plate-based screen needs: plates, runs and acquisition days each carry their own offset, and a single pooled control folds that offset into the estimate.

Usage

```
cr_standardize_batch(
  experiment,
  channel,
  control_level,
  batch_vars,
  control_var = "treatment",
  method = c("log2_fc", "zscore", "raw"),
  eps = 1,
  sd_floor = 1e-08,
  value_to = "value_std",
  on_missing_control = c("error", "warn", "drop")
)
```

Arguments

experiment	A cr_experiment.
channel	Channel column to standardize. Must be numeric.
control_level	Value (or values) of control_var that mark the control cells of a batch.
batch_vars	Character vector of columns that jointly define a batch. See cr_batch_key().
control_var	Column holding the treatment assignment. Defaults to "treatment".
method	Which quantity becomes the canonical analysis value in value_to: "log2_fc" (default), "zscore" (the per-batch z-score z_ctrl), or "raw" (the untransformed channel, for a like-for-like comparison of standardized against measured signal).
eps	Additive offset for the log2 fold change.
sd_floor	Lower bound for the control standard deviation.
value_to	Name of the column receiving the value selected by method. Use NULL to add the component columns only.
on_missing_control	What to do with batches that have no control cells: "error" (default), "warn" or "drop".

Details

Three quantities are always added, whichever method is chosen, because downstream steps need different ones: log2_fc, z_ctrl and the untouched channel. method only decides which of them is copied into value_to as the canonical analysis value.

The log2 fold change uses an *additive* offset, log2((y + eps) / (ctrl_mean + eps)), not a floor on y. An offset keeps the transformation monotone and well behaved near zero, whereas clamping small values to a floor flattens a whole range of the signal onto one number.

The fold change divides by the control **mean** while cr_batch_reference() also reports the control **median**; see there for why both are kept.

A batch that contains no control cells cannot be standardized: its reference does not exist, and a value computed against some other batch's control would be a silent error rather than a measurement. cr_standardize_batch() therefore refuses by default. Use on_missing_control = "drop" to remove those cells (recorded in the QC log) or "warn" to keep them with NA standardized values.

Value

The cr_experiment with these columns added to cells:

batch_key The batch each cell belongs to.

ctrl_n, ctrl_mean, ctrl_median, ctrl_sd The reference statistics of that batch.

z_ctrl (y - ctrl_mean) / ctrl_sd.

log2_fc log2((y + eps) / (ctrl_mean + eps)).

value_to The value selected by method.

The reference table is stored as $batch_reference, the batch columns as $batch_vars, and the settings under $metadata$standardization.

See Also

cr_batch_reference(), cr_batch_key(), cr_normalize()

Other batch standardization functions: cr_batch_key(), cr_batch_reference(), cr_correct_batch()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 20)
exp$design$plate <- rep(c("P1", "P2"), length.out = nrow(exp$design))

std <- cr_standardize_batch(exp, channel = "marker_1",
                            control_level = "Untreated",
                            batch_vars = "plate")
round(summary(std$cells$log2_fc), 2)
std$batch_reference

# Per-batch z-score instead of the fold change
z <- cr_standardize_batch(exp, channel = "marker_1",
                          control_level = "Untreated",
                          batch_vars = "plate", method = "zscore")
round(stats::median(z$cells$value_std), 2)

# A batch without control cells is refused, not guessed at
exp$design$plate[exp$design$treatment == "Untreated"] <- "P1"
try(cr_standardize_batch(exp, channel = "marker_1",
                         control_level = "Untreated",
                         batch_vars = "plate"))
```

`cr_summarize_wells` *Summarize cell-level data to the analysis unit*

Description

Aggregates the per-cell values of one channel to a single number per analysis unit. By default the analysis unit is the spatial unit of the experiment (`well` or `slide`), which is the unit of replication for most plate-based assays. Pass `unit` to aggregate to a merged analysis unit instead — for example when one physical unit was acquired as several files and the acquisitions were assigned a common identifier.

Usage

```
cr_summarize_wells(experiment, channel, fun = stats::median, unit = NULL)
```

Arguments

`experiment`	A `cr_experiment`.
`channel`	Channel name (a column of `experiment$cells`).

fun	Aggregation function. Default stats::median. The function is called with na.rm = TRUE when it accepts that argument (directly or through ...), and with the values only otherwise. It must return a single number.
unit	Optional name of the column that identifies the analysis unit. May be a column of cells or of design. If NULL (default) the experiment's unit_var slot is used when present, and the spatial unit otherwise.

Value

A tibble with one row per analysis unit containing the unit identifier, n_cells, the aggregated value, and the design columns. Design columns that are not constant within a unit are returned as NA.

See Also

cr_compute_metrics() for a richer per-unit summary and cr_table_disposition() for unit and cell counts per arm.

Other quantification: cr_compute_metrics(), cr_fold_change(), cr_table_disposition()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
cr_summarize_wells(exp, channel = "marker_1")

# aggregate with a different estimator
cr_summarize_wells(exp, channel = "marker_1", fun = mean)
```

---

cr_table_disposition *Tabulate how many units and cells entered the analysis*

Description

Counts of analysis units and cells, overall and per arm. The reported counts of an assay are themselves an analysis output: once they are generated from the object that was analyzed, a count in the write-up cannot drift away from the data it describes.

Usage

```
cr_table_disposition(experiment, by = NULL, unit = NULL, total = TRUE)
```

Arguments

experiment	A cr_experiment.
by	Character vector of grouping columns from design or cells. Defaults to "treatment" when that column exists, and to no grouping otherwise.
unit	Optional name of the column that identifies the analysis unit. May be a column of cells or of design. If NULL (default) the experiment's unit_var slot is used when present, and the spatial unit otherwise.

`total` Whether to append a row holding the overall counts. The grouping columns of that row carry the label `"(total)"`.

Value

A tibble with the grouping columns (as character), `n_units`, `n_cells` and `median_cells_per_unit`.

See Also

`cr_tables()`, `cr_macros()`.

Other quantification: `cr_compute_metrics()`, `cr_fold_change()`, `cr_summarize_wells()`

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 20)
cr_table_disposition(exp)
cr_table_disposition(exp, by = "group")
```

---

cr_table_qc *Tabulate the quality-control record*

Description

Normalizes the several shapes a QC record can take — an experiment's QC log, a gate object, or a plain data frame — into one tibble suitable for a supplement.

Usage

```
cr_table_qc(x)
```

Arguments

`x` A `cr_experiment`, a `cr_report`, a data frame, or a list holding data frames (for example a gate object with a `units` element).

Value

A tibble. For a list of several data frames the elements are stacked and identified by a `component` column.

See Also

`cr_tables()`.

Other reporting: `cr_export_plots()`, `cr_export_results()`, `cr_export_tables()`, `cr_render_report()`, `cr_report()`, `cr_tables()`

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 20)
exp <- cr_filter_cells(exp, area > 100)
cr_table_qc(exp)
```

`cr_tables` *Collect the tables of an analysis*

Description

Gathers every tabular artifact of an experiment or report into one named list, ready for `cr_export_tables()`. Empty tables are dropped, so the result reflects what actually exists.

Usage

```
cr_tables(x, results = NULL, which = NULL)
```

Arguments

`x`	A `cr_report` or a `cr_experiment`.
`results`	Optional results to summarize when `x` is a `cr_experiment`. Ignored for a `cr_report`, which carries its own.
`which`	Optional character vector selecting a subset of tables by name. Unknown names are an error that lists what is available.

Value

A named list of tibbles.

See Also

`cr_export_tables()`, `cr_table_disposition()`, `cr_table_qc()`.

Other reporting: `cr_export_plots()`, `cr_export_results()`, `cr_export_tables()`, `cr_render_report()`, `cr_report()`, `cr_table_qc()`

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 20)
tabs <- cr_tables(exp)
names(tabs)
tabs$disposition
```

cr_test *Hypothesis test comparing treatment to control*

Description

Runs a parametric or non-parametric two-sample test comparing cells (or replicate summaries) from a treatment group to the control group on a single channel.

Usage

```
cr_test(
  experiment,
  channel,
  treatment,
  control,
  test = c("mann_whitney", "t_test", "welch", "wilcoxon_signed"),
  level = c("cell", "replicate", "both")
)
```

Arguments

experiment	A cr_experiment.
channel	Channel to test.
treatment	Name of the treatment group (matches design$treatment).
control	Name of the control group.
test	One of "mann_whitney", "t_test" (pooled variance), "welch" (unequal variance). The legacy "wilcoxon_signed" choice is refused: this interface has no explicit matched-pair key. Align complete pairs by a verified identifier and use stats::wilcox.test() with paired = TRUE for paired inference.
level	"cell" (default), "replicate" or "both".

Value

A cr_result object with the elements comparison, cell_level, rep_level, effect_sizes and fold_change. Effect sizes use cells for level = "cell" and replicate medians for "replicate" or "both". With "both", the additional cell_effect_sizes element retains the cell-level comparison. Replicate effect estimates differ from versions through 0.2.0.9000, which computed cell effects regardless of the requested level.

See Also

cr_test_all(), cr_effect_size(), cr_effect_grid().

Other statistics: cr_test_all()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
res <- cr_test(exp,
               channel = "marker_1",
               treatment = "CompoundA_high",
               control = "Untreated",
               test = "mann_whitney",
               level = "replicate")
print(res)
```

---

cr_test_all *Test all treatments against a control group*

Description

Runs `cr_test()` pairwise for every treatment versus the control and adjusts the p-values across the whole family of comparisons. Both a Bonferroni and a Benjamini-Hochberg adjustment are reported next to the unadjusted p-value, never in place of it.

Usage

```
cr_test_all(
  experiment,
  channel,
  control_group,
  tests = "mann_whitney",
  p_adjust = "BH",
  level = c("replicate", "cell", "both")
)
```

Arguments

`experiment`	A `cr_experiment`.
`channel`	Channel name.
`control_group`	Control group.
`tests`	Test to run (see `cr_test()`). Only the first element is used; the argument is a vector for backwards compatibility.
`p_adjust`	P-value adjustment method used for the `p_adj` column (see stats::p.adjust). The dedicated `p_bonferroni` and `p_BH` columns are always added as well.
`level`	`"cell"`, `"replicate"` or `"both"`.

Value

A named list of `cr_result` objects, one per treatment, carrying the attributes `summary` (a tibble with `treatment`, `log2_fc`, `p_value`, `cohens_d`, `p_adj`, `p_bonferroni`, `p_BH` and `interpretation`), `control_group`, `channel` and `level`.

See Also

cr_test(), cr_effect_grid().

Other statistics: cr_test()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 20)
all_res <- cr_test_all(exp, channel = "marker_1",
                       control_group = "Untreated",
                       level = "replicate")
attr(all_res, "summary")
```

---

cr_theme *Publication theme*

Description

The single ggplot2 theme every cr_plot_* function applies, so that figures assembled from different functions sit together without re-styling: a classic axis-line frame, a faint major grid, a bold title, a muted subtitle and a bottom legend.

Usage

```
cr_theme(
  base_size = 11,
  base_family = "sans",
  grid = TRUE,
  legend_position = "bottom",
  variant = c("publication", "report"),
  mode = c("colour", "grayscale")
)
```

Arguments

base_size	Base font size in points (default 11).
base_family	Base font family. Default generic "sans".
grid	Draw the faint major grid (default TRUE).
legend_position	Passed to ggplot2::theme(). Default "bottom".
variant	"publication" or a tighter "report" layout.
mode	Colour or grayscale mode. Typography is shared across modes.

Value

A ggplot2 theme object, usable with + like any other theme.

See Also

cr_palette(), cr_scale_group()

Other plot design: cr_palette(), cr_scale_group(), cr_shapes()

Examples

```
df <- data.frame(g = rep(c("a", "b"), each = 20), y = rnorm(40))
ggplot2::ggplot(df, ggplot2::aes(g, y)) +
  ggplot2::geom_boxplot() +
  cr_theme()
```

---

cr_unit_map *Map source files to analysis units*

Description

Summarises which file contributed how many cells to which unit, and flags the units that were assembled from more than one file. This is the post-condition check for cr_assign_units(): a unit built from several files should be one you can name and justify.

Usage

```
cr_unit_map(
  x,
  id_col = "well_id",
  file_col = "source_path",
  call = rlang::caller_env()
)
```

Arguments

x	A cell table carrying a unit identifier and a file column, as returned by cr_assign_units().
id_col	Name of the unit identifier column. Default "well_id".
file_col	Name of the file column. Default "source_path"; the path rather than the base name, because base names repeat across plates.
call	The execution environment of the calling function. Used for error reporting; experts only.

Value

A tibble with one row per file: the unit identifier, the file, n_cells, n_files (files contributing to that unit) and merged (TRUE when n_files > 1).

See Also

cr_assign_units().

Other import: cr_assign_units(), cr_centroid_overlap(), cr_column_map(), cr_extract_markers(), cr_filename_grammar(), cr_marker_rules(), cr_merge_rules(), cr_parse_paths(), cr_path_spec(), cr_read_cellprofiler(), cr_read_cells(), cr_read_design(), cr_read_export(), cr_read_exports(), cr_read_qupath(), cr_read_segmantr()

Examples

```
cells <- tibble::tibble(
  source_path = rep(c("a.csv", "b.csv", "c.csv"), times = c(3, 2, 4)),
  compound = "CompoundA",
  replicate = rep(c("1", "1.1", "2"), times = c(3, 2, 4)),
  merge_unit = rep(c(FALSE, TRUE, FALSE), times = c(3, 2, 4))
)
units <- cr_assign_units(cells, key_vars = "compound")
cr_unit_map(units)
```

---

cr_unit_variability *Between-unit variability within a condition*

Description

Summarizes how far apart the units of one condition sit. For every group defined by by that holds at least min_units units, the number of units, the coefficient of variation, the spread (max - min of the unit values) and the fold range (log_base ^ spread) are reported.

Usage

```
cr_unit_variability(data, value, by, unit = NULL, min_units = 2, log_base = 2)
```

Arguments

data	A data frame with one row per unit (or per cell when unit is given), or a cr_experiment.
value	Name of the numeric response column, typically a log-scale fold change.
by	Character vector of columns defining a condition, for example compound, experiment, plate and pre-treatment interval.
unit	Optional column identifying the unit of replication. When given, rows are averaged per unit first.
min_units	Minimum number of units a group must contain to be reported (default 2).
log_base	Base of the logarithm the response is on. Used to turn the spread into a fold range. Default 2.

Details

This is unit-to-unit variability. It is **not** within-unit technical repeatability: that would require the same unit to be measured twice, which a design with one acquisition per unit cannot deliver. A unit assembled from two partial acquisitions is two halves of one unit, not two reads of it.

The coefficient of variation is computed as `100 * sd / abs(mean)`, which is unstable when the mean of a log-scale response approaches zero; the fold range is the robust summary and is what the attached overall summary reports.

Value

A tibble with the by columns, `n_units`, `mean_value`, `sd_value`, `cv_pct`, `spread` and `fold_range`, carrying a `summary` attribute (a one-row tibble with the number of groups and the median, quartiles and maximum of the fold range).

See Also

`cr_blocked_effect()`, `cr_effect_grid()`.

Other screen statistics: `cr_blocked_effect()`

Examples

```
set.seed(1)
units <- data.frame(
  compound = rep(c("CompoundA", "CompoundB"), each = 12),
  plate = rep(rep(c("P1", "P2"), each = 6), 2),
  log2_fc = stats::rnorm(24, 0, 0.4)
)
v <- cr_unit_variability(units, value = "log2_fc",
                         by = c("compound", "plate"))
v
attr(v, "summary")
```

---

cr_validate_experiment

Validate a `cr_experiment`

Description

Performs structural checks on a `cr_experiment`. Called automatically by `cr_build_experiment()` but can also be used to verify that manual modifications have not broken the object.

Usage

```
cr_validate_experiment(x, call = rlang::caller_env())
```

Arguments

x	A cr_experiment.
call	The execution environment of the calling function. Used for error reporting; experts only.

Details

The optional unit_var, batch_vars, provenance and set_aside slots are checked only when they are present, so that objects built by earlier versions still validate.

Value

TRUE invisibly on success. On failure an informative error is raised.

See Also

cr_build_experiment().

Other constructors: cr_build_experiment(), cr_dataset(), cr_design()

Examples

```
cells <- tibble::tibble(
  cell_id = c("c1", "c2"), well = c("A01", "A02"),
  target_signal = c(10, 20)
)
design <- tibble::tibble(well = c("A01", "A02"),
                         treatment = c("Vehicle", "CompoundA"))
cr_validate_experiment(cr_build_experiment(cells, design))
```

---

cr_validate_report_evidence

Revalidate report evidence against its current source tree

Description

Revalidate report evidence against its current source tree

Usage

```
cr_validate_report_evidence(evidence, root)
```

Arguments

evidence	A cr_report_evidence() object.
root	Current root of the source tree, including after relocation.

Value

Invisibly TRUE; invalid schemas, changed bytes or altered claims error.

cr_validate_report_qc *Validate a concise report QC table*

Description

Validate a concise report QC table

Usage

```
cr_validate_report_qc(qc)
```

Arguments

qc	A data frame with criterion, observed, acceptance and status columns.

cr_validate_report_spec *Validate a laboratory report specification*

Description

Validate a laboratory report specification

Usage

```
cr_validate_report_spec(spec, strict = TRUE, return_issues = FALSE)
```

Arguments

spec	A cr_report_spec.
strict	In strict mode structural minima and caller-defined required fields are errors.
return_issues	Return a severity-coded issue table instead of throwing errors.

Value

TRUE invisibly, or a data frame when return_issues = TRUE.

See Also

Other laboratory reporting: cr_report_display_data(), cr_report_profile(), cr_report_spec(), cr_report_style()

cr_well_to_rowcol *Convert well IDs to row and column indices*

Description

Splits well identifiers of the form "A01" into the letter row and numeric column.

Usage

```
cr_well_to_rowcol(well)
```

Arguments

well	A character vector of well IDs (e.g. "A01", "H12", "P24").

Value

A tibble with columns well, row (integer, 1 = A) and col (integer).

Examples

```
cr_well_to_rowcol(c("A01", "B05", "H12"))
```

print.cr_experiment *Print method for* cr_experiment

Description

Print method for cr_experiment

Usage

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

Arguments

x	A cr_experiment.
...	Unused.

Value

x invisibly.

print.cr_qc_gate *Print a QC gate*

Description

Print a QC gate

Usage

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

Arguments

x	A cr_qc_gate from cr_qc_gate().
...	Ignored.

Value

x, invisibly.

See Also

Other quality control: cr_apply_gate(), cr_balance_cells(), cr_exclude_small(), cr_qc_doublets(), cr_qc_filter(), cr_qc_gate(), cr_qc_gate_impact(), cr_qc_intensity(), cr_qc_manual(), cr_qc_report(), cr_qc_summary()

Examples

```
exp <- cr_example_experiment(seed = 1, n_cells_per_well = 30)
gate <- cr_qc_gate(exp, "marker_1", "Untreated", batch_vars = "replicate")
print(gate)
```

print.cr_report *Print method for* cr_report

Description

Print method for cr_report

Usage

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

Arguments

x	A cr_report.
...	Unused.

Value

x invisibly.

---

print.cr_result *Print method for* cr_result

Description

Print method for cr_result

Usage

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

Arguments

x	A cr_result.
...	Unused.

Value

x invisibly.

---

summary.cr_experiment *Summary method for* cr_experiment

Description

Summary method for cr_experiment

Usage

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

Arguments

object	A cr_experiment.
...	Unused.

Value

A tibble summarising the experiment (invisibly printed).

---

summary.cr_report *Summary method for* cr_report

Description

Summary method for cr_report

Usage

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

Arguments

object	A cr_report.
...	Unused.

Value

object invisibly.

---

summary.cr_result *Summary method for* cr_result

Description

Summary method for cr_result

Usage

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

Arguments

object	A cr_result.
...	Unused.

Value

The result list, invisibly.

Index

∗ **batch standardization functions**
cr_batch_key, 10
cr_batch_reference, 11
cr_correct_batch, 23
cr_standardize_batch, 133
∗ **cellreportR-app**
cr_run_app, 127
∗ **constructors**
cr_build_experiment, 14
cr_dataset, 25
cr_design, 26
cr_validate_experiment, 144
∗ **effect sizes**
cr_compare_levels, 20
cr_effect_grid, 29
cr_effect_size, 31
∗ **example data**
cr_example_design, 33
cr_example_experiment, 34
cr_example_exports, 36
cr_example_files, 37
cr_example_path, 38
cr_example_screen, 39
∗ **import**
cr_assign_units, 6
cr_centroid_overlap, 16
cr_column_map, 19
cr_extract_markers, 46
cr_filename_grammar, 48
cr_marker_rules, 62
cr_merge_rules, 65
cr_parse_paths, 69
cr_path_spec, 71
cr_read_cellprofiler, 105
cr_read_cells, 106
cr_read_design, 107
cr_read_export, 108
cr_read_exports, 109
cr_read_qupath, 111
cr_read_segmantr, 112
cr_unit_map, 142
∗ **laboratory reporting**
cr_report_display_data, 117
cr_report_profile, 119
cr_report_spec, 123
cr_report_style, 124
cr_validate_report_spec, 146
∗ **macros**
cr_enumerate, 32
cr_format_number, 53
cr_macro_name, 59
cr_macros, 60
cr_macros_from, 61
∗ **normalization functions**
cr_background_subtract, 8
cr_normalize, 67
∗ **plot design**
cr_palette, 68
cr_scale_group, 131
cr_shapes, 132
cr_theme, 141
∗ **power**
cr_conservative_effect, 23
cr_power, 93
cr_power_analysis, 94
cr_power_grid, 95
∗ **quality control**
cr_apply_gate, 5
cr_balance_cells, 9
cr_exclude_small, 41
cr_qc_doublets, 97
cr_qc_filter, 98
cr_qc_gate, 99
cr_qc_gate_impact, 101
cr_qc_intensity, 102
cr_qc_manual, 103
cr_qc_report, 104
cr_qc_summary, 105

print.cr_qc_gate, 148
∗ **quantification**
cr_compute_metrics, 21
cr_fold_change, 51
cr_summarize_wells, 135
cr_table_disposition, 136
∗ **reporting**
cr_export_plots, 42
cr_export_results, 44
cr_export_tables, 45
cr_render_report, 114
cr_report, 115
cr_table_qc, 137
cr_tables, 138
∗ **screen figures**
cr_plot_forest, 76
cr_plot_qc_gate, 81
cr_plot_sample_size, 85
cr_plot_screen, 87
cr_plot_specificity, 90
cr_save_plot, 129
∗ **screen statistics**
cr_blocked_effect, 13
cr_unit_variability, 143
∗ **statistics**
cr_test, 139
cr_test_all, 140
∗ **visualisation**
cr_plot_comparison, 72
cr_plot_dose_response, 73
cr_plot_effect_sizes, 73
cr_plot_foldchange, 75
cr_plot_heatmap, 77
cr_plot_histogram, 78
cr_plot_intensity, 79
cr_plot_plate, 80
cr_plot_qc, 81
cr_plot_roc, 84
cr_plot_scatter, 86
cr_plot_spatial, 89
cr_plot_timeline, 92

cr_apply_gate, 5
cr_apply_gate(), *10*, *42*, *97, 98*, *100*, *102–105*, *148*
cr_assign_units, 6
cr_assign_units(), *17*, *20*, *47*, *49*, *63*, *66*, *70, 71*, *106, 107*, *109*, *111, 112*, *142, 143*
cr_auc, 7
cr_background_subtract, 8
cr_background_subtract(), *67, 68*
cr_balance_cells, 9
cr_balance_cells(), *6*, *42*, *97, 98*, *100*, *102–105*, *148*
cr_batch_key, 10
cr_batch_key(), *12*, *24*, *134, 135*
cr_batch_reference, 11
cr_batch_reference(), *11*, *24*, *39*, *41*, *134, 135*
cr_blocked_effect, 13
cr_blocked_effect(), *31*, *39*, *144*
cr_build_experiment, 14
cr_build_experiment(), *25, 26*, *28*, *128*, *144, 145*
cr_centroid_overlap, 16
cr_centroid_overlap(), *7*, *20*, *47*, *49*, *63*, *66*, *70, 71*, *106, 107*, *109*, *111, 112*, *143*
cr_channels, 17
cr_classify_result, 18
cr_column_map, 19
cr_column_map(), *7*, *17*, *36*, *38*, *47*, *49*, *63*, *66*, *70, 71*, *106–112*, *143*
cr_compare_levels, 20
cr_compare_levels(), *30–32*
cr_compute_metrics, 21
cr_compute_metrics(), *52*, *136, 137*
cr_confusion_matrix, 22
cr_conservative_effect, 23
cr_conservative_effect(), *94–96*
cr_correct_batch, 23
cr_correct_batch(), *11, 12*, *68*, *135*
cr_dataset, 25
cr_dataset(), *15, 16*, *28*, *111*, *145*
cr_design, 26
cr_design(), *15, 16*, *25, 26*, *107*, *145*
cr_dose_response, 28
cr_dose_response(), *55*
cr_effect_grid, 29
cr_effect_grid(), *14*, *20, 21*, *23*, *32*, *39*, *41*, *95, 96*, *139*, *141*, *144*
cr_effect_size, 31
cr_effect_size(), *21*, *29–31*, *139*
cr_enumerate, 32
cr_enumerate(), *54*, *59*, *61, 62*
cr_example_design, 33

cr_example_design(), *35–38*, *41*
cr_example_experiment, 34
cr_example_experiment(), *33*, *34*, *36–39*, *41*
cr_example_exports, 36
cr_example_exports(), *34*, *35*, *37*, *38*, *41*
cr_example_files, 37
cr_example_files(), *34–36*, *38*, *41*
cr_example_path, 38
cr_example_path(), *34–37*, *41*
cr_example_screen, 39
cr_example_screen(), *34–38*
cr_exclude_small, 41
cr_exclude_small(), *6*, *10*, *40*, *97*, *98*, *100*, *102–105*, *148*
cr_export_plots, 42
cr_export_plots(), *45*, *114*, *116*, *137*, *138*
cr_export_report_audit, 43
cr_export_report_spec, 44
cr_export_results, 44
cr_export_results(), *43*, *45*, *114*, *116*, *137*, *138*
cr_export_tables, 45
cr_export_tables(), *43*, *45*, *114*, *116*, *137*, *138*
cr_extract_markers, 46
cr_extract_markers(), *7*, *17*, *20*, *36*, *49*, *63*, *66*, *70*, *71*, *106*, *107*, *109*, *111*, *112*, *143*
cr_figure_caption, 47
cr_figure_size, 48
cr_filename_grammar, 48
cr_filename_grammar(), *7*, *17*, *20*, *47*, *63*, *66*, *70*, *71*, *106*, *107*, *109*, *111*, *112*, *143*
cr_filter_cells, 50
cr_finalize_result, 51
cr_fold_change, 51
cr_fold_change(), *22*, *136*, *137*
cr_format_interval, 53
cr_format_number, 53
cr_format_number(), *33*, *59*, *61*, *62*, *118*
cr_format_percent, 54
cr_format_pvalue, 54
cr_ic50, 55
cr_import_report_spec, 55
cr_interpretation_lookup, 56
cr_lab_report, 56
cr_lab_report(), *57*
cr_lab_report_from_evidence, 57
cr_lab_report_from_evidence(), *122*
cr_linetype_scale *(*cr_scale_colour*)*, 130
cr_linetypes, 58
cr_logistic, 58
cr_logistic(), *8*, *22*, *126*
cr_macro_name, 59
cr_macro_name(), *33*, *54*, *61*, *62*
cr_macros, 60
cr_macros(), *33*, *45*, *54*, *59*, *61*, *62*, *116*, *137*
cr_macros_from, 61
cr_macros_from(), *33*, *54*, *59*, *61*
cr_marker_rules, 62
cr_marker_rules(), *7*, *17*, *20*, *46*, *47*, *49*, *66*, *70*, *71*, *106*, *107*, *109*, *111*, *112*, *143*
cr_merge_experiments, 64
cr_merge_rules, 65
cr_merge_rules(), *6*, *7*, *17*, *20*, *47*, *49*, *63*, *70*, *71*, *106*, *107*, *109*, *111*, *112*, *143*
cr_n_cells, 66
cr_normalize, 67
cr_normalize(), *9*, *135*
cr_palette, 68
cr_palette(), *80*, *131–133*, *142*
cr_parse_paths, 69
cr_parse_paths(), *7*, *17*, *20*, *36*, *47*, *49*, *63*, *66*, *71*, *106*, *107*, *109–112*, *143*
cr_path_spec, 71
cr_path_spec(), *7*, *17*, *20*, *36*, *47*, *49*, *63*, *66*, *70*, *106*, *107*, *109–112*, *143*
cr_plot_comparison, 72
cr_plot_comparison(), *73–75*, *78–81*, *85*, *87*, *89*, *92*
cr_plot_dose_response, 73
cr_plot_dose_response(), *72*, *74*, *75*, *78–81*, *85*, *87*, *89*, *92*
cr_plot_effect_sizes, 73
cr_plot_effect_sizes(), *72*, *73*, *75*, *77–81*, *85*, *87*, *89*, *92*
cr_plot_estimates, 74
cr_plot_foldchange, 75
cr_plot_foldchange(), *72–74*, *78–81*, *85*, *87*, *89*, *92*
cr_plot_forest, 76
cr_plot_forest(), *73–75*, *83*, *86*, *89*, *91*, *130*
cr_plot_heatmap, 77
cr_plot_heatmap(), *72–75*, *79–81*, *85*, *87*,

89, *92*
cr_plot_histogram, 78
cr_plot_histogram(), *72–75*, *78*, *80, 81*, *85*, *87*, *89*, *92*
cr_plot_intensity, 79
cr_plot_intensity(), *72–75*, *78–81*, *85*, *87*, *89*, *92*
cr_plot_plate, 80
cr_plot_plate(), *72–75*, *78–81*, *85*, *87*, *89*, *92*
cr_plot_qc, 81
cr_plot_qc(), *72–75*, *78–80*, *83*, *85*, *87*, *89*, *92*
cr_plot_qc_gate, 81
cr_plot_qc_gate(), *77*, *81*, *86*, *89*, *91*, *130*
cr_plot_qc_summary, 83
cr_plot_result_position, 83
cr_plot_roc, 84
cr_plot_roc(), *72–75*, *78–81*, *87*, *89*, *92*
cr_plot_sample_size, 85
cr_plot_sample_size(), *77*, *83*, *89*, *91*, *130*
cr_plot_scatter, 86
cr_plot_scatter(), *72–75*, *78–81*, *85*, *89*, *92*
cr_plot_screen, 87
cr_plot_screen(), *77*, *80*, *83*, *86*, *91*, *130*
cr_plot_spatial, 89
cr_plot_spatial(), *72–75*, *78–81*, *85*, *87*, *92*
cr_plot_specificity, 90
cr_plot_specificity(), *77*, *83*, *86*, *89*, *130*
cr_plot_style, 91
cr_plot_style(), *84*
cr_plot_timeline, 92
cr_plot_timeline(), *72–75*, *78–81*, *85*, *87*, *89*
cr_power, 93
cr_power(), *23*, *32*, *94–96*
cr_power_analysis, 94
cr_power_analysis(), *23*, *94*, *96*
cr_power_grid, 95
cr_power_grid(), *23*, *31*, *94, 95*
cr_print_report_spec, 96
cr_qc_doublets, 97
cr_qc_doublets(), *6*, *10*, *42*, *98*, *100*, *102–105*, *148*
cr_qc_filter, 98
cr_qc_filter(), *6*, *10*, *41, 42*, *97*, *100*, *102–105*, *148*
cr_qc_gate, 99
cr_qc_gate(), *5, 6*, *10*, *39*, *42*, *97, 98*, *101–105*, *148*
cr_qc_gate_impact, 101
cr_qc_gate_impact(), *6*, *10*, *42*, *97, 98*, *100*, *103–105*, *148*
cr_qc_intensity, 102
cr_qc_intensity(), *6*, *10*, *42*, *97, 98*, *100*, *102–105*, *148*
cr_qc_manual, 103
cr_qc_manual(), *6*, *10*, *42*, *97, 98*, *100*, *102–105*, *148*
cr_qc_report, 104
cr_qc_report(), *6*, *10*, *42*, *97, 98*, *100*, *102, 103*, *105*, *148*
cr_qc_summary, 105
cr_qc_summary(), *6*, *10*, *42*, *97, 98*, *100*, *102–104*, *148*
cr_read_cellprofiler, 105
cr_read_cellprofiler(), *7*, *17*, *20*, *47*, *49*, *63*, *66*, *70, 71*, *107*, *109*, *111, 112*, *143*
cr_read_cells, 106
cr_read_cells(), *7*, *17*, *20*, *47*, *49*, *63*, *66*, *70, 71*, *106, 107*, *109*, *111, 112*, *143*
cr_read_design, 107
cr_read_design(), *7*, *17*, *20*, *28*, *38*, *47*, *49*, *63*, *66*, *70, 71*, *106, 107*, *109*, *111, 112*, *143*
cr_read_export, 108
cr_read_export(), *7*, *17*, *20*, *38*, *47*, *49*, *63*, *66*, *70, 71*, *106, 107*, *109–112*, *143*
cr_read_exports, 109
cr_read_exports(), *7*, *17*, *20*, *25, 26*, *36*, *47*, *49*, *63*, *66*, *70, 71*, *106, 107*, *109*, *112*, *143*
cr_read_qupath, 111
cr_read_qupath(), *7*, *17*, *20*, *47*, *49*, *63*, *66*, *70, 71*, *106, 107*, *109*, *111, 112*, *143*
cr_read_segmantr, 112
cr_read_segmantr(), *7*, *17*, *20*, *47*, *49*, *63*, *66*, *70, 71*, *106, 107*, *109*, *111, 112*, *143*
cr_render_lab_report, 113
cr_render_lab_report(), *122*
cr_render_report, 114
cr_render_report(), *43*, *45*, *116*, *137, 138*

cr_report, 115
cr_report(), *43*, *45*, *114*, *128*, *137, 138*
cr_report_data, 117
cr_report_display_data, 117
cr_report_display_data(), *120*, *124, 125*, *146*
cr_report_evidence, 118
cr_report_evidence(), *57*, *122*, *145*
cr_report_hash, 119
cr_report_profile, 119
cr_report_profile(), *57*, *118*, *124, 125*, *127*, *146*
cr_report_provenance, 120
cr_report_qc, 121
cr_report_qc(), *56*
cr_report_qc_from_log, 121
cr_report_review_app, 122
cr_report_spec, 123
cr_report_spec(), *57*, *118*, *120*, *125*, *127*, *146*
cr_report_style, 124
cr_report_style(), *57*, *118*, *120*, *124*, *146*
cr_roc, 126
cr_rowcol_to_well, 126
cr_run_app, 127
cr_save_figure, 128
cr_save_plot, 129
cr_save_plot(), *77*, *83*, *86*, *89*, *91*
cr_scale_colour, 130
cr_scale_diverging, 131
cr_scale_fill *(*cr_scale_colour*)*, 130
cr_scale_group, 131
cr_scale_group(), *69*, *89*, *133*, *142*
cr_shape_scale *(*cr_scale_colour*)*, 130
cr_shapes, 132
cr_shapes(), *69*, *131, 132*, *142*
cr_standardize_batch, 133
cr_standardize_batch(), *9*, *11, 12*, *24*, *39*, *68*
cr_summarize_wells, 135
cr_summarize_wells(), *22*, *52*, *137*
cr_table_disposition, 136
cr_table_disposition(), *22*, *52*, *136*, *138*
cr_table_qc, 137
cr_table_qc(), *43*, *45*, *114*, *116*, *138*
cr_tables, 138
cr_tables(), *43*, *45*, *114*, *116*, *137*
cr_test, 139
cr_test(), *73*, *140, 141*
cr_test_all, 140
cr_test_all(), *44*, *73–75*, *115*, *139*
cr_theme, 141
cr_theme(), *69*, *80*, *89*, *132, 133*
cr_unit_map, 142
cr_unit_map(), *7*, *17*, *20*, *47*, *49*, *63*, *66*, *70, 71*, *106, 107*, *109*, *111, 112*
cr_unit_variability, 143
cr_unit_variability(), *14*
cr_validate_experiment, 144
cr_validate_experiment(), *16*, *26*, *28*, *127*
cr_validate_report_evidence, 145
cr_validate_report_qc, 146
cr_validate_report_spec, 146
cr_validate_report_spec(), *118*, *120*, *124, 125*
cr_well_to_rowcol, 147

digest::digest(), *119*

ggplot2::ggsave(), *130*
ggplot2::scale_shape_manual(), *133*
ggplot2::theme(), *141*

print.cr_column_map *(*cr_column_map*)*, 19
print.cr_dataset *(*cr_dataset*)*, 25
print.cr_design *(*cr_design*)*, 26
print.cr_experiment, 147
print.cr_filename_grammar *(*cr_filename_grammar*)*, 48
print.cr_merge_rules *(*cr_merge_rules*)*, 65
print.cr_path_spec *(*cr_path_spec*)*, 71
print.cr_qc_gate, 148
print.cr_qc_gate(), *6*, *10*, *42*, *97, 98*, *100*, *102–105*
print.cr_report, 148
print.cr_result, 149

rmarkdown::render(), *113*

shiny::runApp(), *127, 128*
shiny::testServer(), *128*
stats::p.adjust, *140*
stats::p.adjust(), *30*
stats::power.t.test(), *93*
stats::wilcox.test(), *139*
summary.cr_dataset *(*cr_dataset*)*, 25

summary.cr_experiment, 149
summary.cr_report, 150
summary.cr_result, 150