Package: qviewparsR (via r-universe)

July 23, 2026

Title Read .Q-View Multiplex ELISA Project Files

Version 1.0.0

Description Pure-R parser for the binary '.Q-View' project file format used in chemiluminescent multiplex ELISA plate imaging and quantification. Reads the embedded H2 database container and CSV report, and returns project metadata, the analyte panel with units and detection limits, sample well-group assignments, per-well pixel-intensity replicates, summary statistics, optional back-calculated concentrations, and a plate layout, all as tidy tibbles. No Java runtime or H2 database driver is required.

License MIT + file LICENSE

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

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

Depends R (>= 4.1.0)

Imports cli, dplyr, lifecycle, openxlsx2, readr, rlang, stats, tibble, tidyr, tools, utils

Suggests bslib, chromote, DT, ggplot2, knitr, patchwork, rmarkdown, shiny, shinytest2, testthat (>= 3.0.0), withr

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Config/roxygen2/version 8.0.0

Config/Needs/website CTTIR/themakR

NeedsCompilation no

Author R. Heller [aut, cre, cph] (ORCID: <https://orcid.org/0000-0001-8006-9742>), M. Mannes [aut, cph] (ORCID: <https://orcid.org/0009-0003-4875-8275>)

Maintainer R. Heller <raban.heller@uni-ulm.de>

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

Date/Publication 2026-07-23 14:39:49 UTC

RemoteUrl https://github.com/cran/qviewparsR

RemoteRef HEAD

RemoteSha 9bbed13b400eb8438311fcb9c315c729b345d722

Contents

`as_tibble.qview` *Coerce a Q-View object to a tibble*

Description

Returns the long-format `pixel_intensities` table — the primary tabular data carried by a `qview` object. To access other slots (well groups, analyte panel, summary statistics) use `qv$<slot>` directly.

Usage

```
## S3 method for class 'qview'
as_tibble(x, ...)
```

Arguments

`x`	A `qview` object returned by `read_qview()`.
`...`	Unused; for S3 generic compatibility.

Value

A `tibble::tibble()` of replicate pixel-intensity readings.

See Also

Other qview-methods: `is_qview()`, `plot.qview()`, `print.qview()`, `print.qview_summary()`, `summary.qview()`

Examples

```
path <- system.file("extdata", "example.Q-View", package = "qviewparsR")
qv <- read_qview(path, verbose = FALSE)
tibble::as_tibble(qv)
```

---

is_qview *Test whether an object is a* qview

Description

[Experimental]

Usage

```
is_qview(x)
```

Arguments

x	An object to test.

Value

Logical scalar.

See Also

Other qview-methods: `as_tibble.qview()`, `plot.qview()`, `print.qview()`, `print.qview_summary()`, `summary.qview()`

Examples

```
path <- system.file("extdata", "example.Q-View", package = "qviewparsR")
is_qview(read_qview(path, verbose = FALSE))
is_qview(list())
```

`plot.qview` *Plot a Q-View object*

Description

Quick-look plots for a parsed `qview` object.

Usage

```
## S3 method for class 'qview'
plot(x, type = c("plate_map", "intensity_heatmap", "replicate_scatter"), ...)
```

Arguments

`x`	A `qview` object returned by `read_qview()`.
`type`	One of `"plate_map"`, `"intensity_heatmap"`, `"replicate_scatter"`. Default `"plate_map"`.
`...`	Unused; for S3 generic compatibility.

Details

- `"plate_map"` – heat-coloured plate map, one cell per well, fill by well type (standard / negative / sample / control).
- `"intensity_heatmap"` – per-analyte facet, fill by replicate-1 pixel intensity per well.
- `"replicate_scatter"` – replicate 1 vs replicate 2 pixel intensity per analyte.

Requires the `ggplot2` package (Suggested).

Value

A `ggplot` object.

See Also

Other qview-methods: `as_tibble.qview()`, `is_qview()`, `print.qview()`, `print.qview_summary()`, `summary.qview()`

Examples

```
path <- system.file("extdata", "example.Q-View", package = "qviewparsR")
qv <- read_qview(path, verbose = FALSE)
plot(qv, type = "plate_map")
plot(qv, type = "replicate_scatter")
```

print.qview *Print a Q-View object*

Description

Compact summary of a parsed `qview` object: project / plate identifiers, analyte panel, well-group counts by type, and whether the embedded report carries quantitative concentrations or only qualitative pixel intensities.

Usage

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

Arguments

`x`	A `qview` object returned by `read_qview()`.
`...`	Ignored.

Value

`x`, invisibly.

See Also

Other qview-methods: `as_tibble.qview()`, `is_qview()`, `plot.qview()`, `print.qview_summary()`, `summary.qview()`

Examples

```
qv <- read_qview(system.file("extdata", "example.Q-View",
                             package = "qviewparsR"), verbose = FALSE)
print(qv)
```

print.qview_summary *Print a qview_summary object*

Description

[Experimental]

Usage

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

Arguments

x	A qview_summary object returned by summary.qview().
...	Unused; for S3 generic compatibility.

Value

x, invisibly.

See Also

Other qview-methods: as_tibble.qview(), is_qview(), plot.qview(), print.qview(), summary.qview()

Examples

```
path <- system.file("extdata", "example.Q-View", package = "qviewparsR")
qv <- read_qview(path, verbose = FALSE)
summary(qv)
```

---

qview_app *Launch the qviewparsR Q-View Shiny app*

Description

[Experimental]

Usage

```
qview_app(max_upload_mb = 512, ...)
```

Arguments

max_upload_mb	Numeric. Maximum upload size per request, in megabytes. Q-View project files routinely exceed the Shiny upload default of 5 MB; this argument bumps the limit for the duration of the running app and restores the previous value on exit. Default 512 MB.
...	Forwarded to shiny::runApp().

Details

Interactive front-end for read_qview(). Uploads a .Q-View file (and optionally an accompanying well-assignment template CSV), displays the parsed metadata, analytes, well groups, and replicate tables, and lets the user download the parsed result as xlsx, rds, or a zip of per-table CSV files.

Requires the shiny, bslib, and DT packages (listed under Suggests).

Value

Invoked for its side effect of running the app.

Examples

```
if (interactive()) {
  qview_app()
}
```

`read_qview` *Read a .Q-View project file*

Description

[Experimental]

Usage

```
read_qview(
  path,
  strip_prefix = FALSE,
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

`path`	Character. Path to the `.Q-View` file.
`strip_prefix`	Logical. If `TRUE`, reverse Q-View's internal naming convention via `strip_qview_prefix()` so identifiers match the original well-assignment template. Default `FALSE`.
`verbose`	Logical. Print a short summary after parsing. Default `TRUE`.
`call`	The execution environment of the calling function. Used for error reporting; experts only.

Details

Parses a `.Q-View` binary container (a chemiluminescent multiplex ELISA project file holding an embedded H2 database plus binary LOB segments) and extracts the assay data: project metadata, analyte panel with units, well-group sample assignments, per-well replicate pixel intensities, summary statistics, and (when present) the embedded CSV report.

The file format is reverse-engineered from public binary inspection: it begins with a plain-text manifest, followed by three concatenated H2 database segments. The fully-formatted report Q-View renders for the user is stored as a CLOB inside the main H2 segment. This parser scans the binary for that CLOB, reassembles it across H2 page boundaries (2048-byte pages), and parses it as CSV.

Parsing is done in pure R: no Java runtime, no H2 database driver, no system dependencies beyond a working R installation.

Value

A list with class "qview" containing:

`metadata` Named list: `project`, `plate`, `image`, `imager`, `product`, `user`, `report_created`, `qview_version`, `template`, `container_version`, `file_path`, `parsed_at`.

`manifest` Tibble with one row per declared file entry (`name`, `size_bytes`, `parent`).

`segments` Tibble of H2 segment byte ranges (`segment`, `start`, `end`, `size`).

`analytes` Tibble: `spot_number`, `analyte`, `unit`, plus `lod`, `lloq`, `uloq`, `assay_control_low`, `assay_control_high` when reported.

`well_groups` Tibble: `well_group`, `sample_id`, `is_standard`, `is_negative`, `is_sample`, `is_control`, `well_type`.

`pixel_intensities` Long-format tibble of replicate readings: `well_group`, `sample_id`, `well`, `replicate`, `analyte`, `unit`, `pixel_intensity`, `dilution`.

`summary_statistics` Long-format tibble of per-group averages, std-dev, and CV statistics: `well_group`, `sample_id`, `statistic`, `analyte`, `value`, `unit`.

`concentrations` Long-format concentration tibble or `NULL` if the regression model is "`Qualitative`".

`curve_fit` Tibble with `analyte`, `regression_model`, or `NULL` if not reported.

`report_csv` Character vector of the raw CSV report lines, or `NULL` if no report was generated.

`plate_layout` Tibble with one row per well: `well`, `plate_row`, `plate_col`, `well_group`, `sample_id`, `well_type`, `dilution`.

See Also

`strip_qview_prefix()`, `read_qview_template()`, `print.qview()`, `plot.qview()`.

Other qview-reader: `read_qview_report()`, `read_qview_template()`

Examples

```
# A small synthetic .Q-View ships with the package:
path <- system.file("extdata", "example.Q-View", package = "qviewparsR")
qv <- read_qview(path)
qv
qv$analytes
head(qv$pixel_intensities)

if (requireNamespace("ggplot2", quietly = TRUE)) {
  plot(qv, type = "plate_map")
}
```

`read_qview_report` *Read a Q-View report export (CSV or XLSX)*

Description

[Experimental]

Usage

```
read_qview_report(
  path,
  strip_prefix = FALSE,
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

`path`	Character. Path to a Q-View report export (`.csv` or `.xlsx`).
`strip_prefix`	Logical. If `TRUE`, reverse Q-View's internal naming convention via `strip_qview_prefix()`. Default `FALSE`.
`verbose`	Logical. Print a short summary after parsing. Default `TRUE`.
`call`	The execution environment of the calling function. Used for error reporting; experts only.

Details

Parses one of the flat report files Q-View exports next to the native `.Q-View` container – the `..._auto_report` or `..._auto_all-parameters_report` export, as either `.csv` or `.xlsx` – and returns the same `qview` object `read_qview()` builds from the binary container. Use it when only the exports were kept and the original `.Q-View` project file is unavailable.

The export shares its report layout with the CLOB embedded in the binary container, so the two readers agree on concentrations and pixel intensities. Two behaviours differ from `read_qview()`, both to preserve information the study workflows rely on:

- The plain `"Reduced Concentration"` point estimate (one row per sample, the value the exports headline) is captured with `statistic == "reduced"`. `read_qview()` currently keeps only the per-replicate / summary concentration rows.
- Out-of-range cells are **preserved, not dropped**: a `"< 52.50"` cell yields `concentration = 52.50` with `flag = "<"`, a `"> 7700"` cell `flag = ">"`, and an `"Incalculable ..."` cell `concentration = NA` with `flag = "incalculable"`. In-range cells carry `flag = NA`.

Value

A list with class "qview", structured as the read_qview() return value, with these deviations. Container-only slots (manifest, segments) are zero-row tibbles; metadata$container_version is NA. The concentrations tibble carries one extra column, flag (NA / "<" / ">" / "incalculable"), relative to read_qview(). report_csv echoes the full export in file order (metadata preamble, blank spacer rows, the analyte header, then the data rows), whereas read_qview()'s report_csv holds only the unique report data lines.

See Also

read_qview(), read_qview_template().

Other qview-reader: read_qview(), read_qview_template()

Examples

```
path <- system.file("extdata", "example-report.csv",
                    package = "qviewparsR")
if (nzchar(path)) {
  qv <- read_qview_report(path)
  qv$concentrations
}
```

---

read_qview_template *Read a Q-View well-assignment template CSV*

Description

[Experimental]

Usage

```
read_qview_template(path, verbose = TRUE, call = rlang::caller_env())
```

Arguments

path	Path to the template file (csv).
verbose	Logical. Print a short summary after parsing. Default TRUE.
call	The execution environment of the calling function. Used for error reporting; experts only.

Details

Parses the plate-template CSV that Q-View imports for sample assignment. The file uses a multi-section layout: an `NxM` cell in the top-left declares the plate dimensions, followed by sections labelled `Group Name`, `Group Type`, and `Dilution Factor`. Each section is one row per plate row and one column per plate column.

All template data is also embedded inside the `.Q-View` file itself (and is recovered by `read_qview()`); this function exists for setting up new plates or cross-validating Q-View imports against the original template.

Value

A `tibble::tibble()` with one row per well and columns `well` (character, e.g. `"A1"`), `plate_row` (character, `"A"`..), `plate_col` (integer), `sample_id` (character), `group_type` (character), `dilution` (numeric).

See Also

Other qview-reader: `read_qview()`, `read_qview_report()`

Examples

```
path <- system.file("extdata", "example-template.csv",
                    package = "qviewparsR")
layout <- read_qview_template(path)
head(layout)
```

---

strip_qview_prefix *Reverse the Q-View internal naming convention*

Description

[Experimental]

Usage

```
strip_qview_prefix(x)
```

Arguments

`x`	Character vector of Q-View internal names.

Details

On import, the producing software prefixes well-assignment template names with single-letter codes:

- "Cal 1" ... "Cal N" -> "ICal 1" ... "ICal N" (Internal calibrator)
- "Low" -> "GLow"
- "High" -> "HHigh"
- "FD..." and any all-digit ID -> "N..." (sample prefix)

strip_qview_prefix() reverses this transformation so that downstream code sees the identifiers exactly as they appear in the original template CSV. Strings that do not match a known prefix pattern are returned unchanged.

Value

Character vector of the same length as x, with prefixes stripped. NAs and unrecognised values pass through unchanged.

See Also

Other qview-helper: well_label()

Examples

```
strip_qview_prefix(c("ICal 1", "GLow", "HHigh",
                     "NFD24277364", "N1211498458", "Plate 1"))
```

---

summary.qview *Summary statistics for a Q-View object*

Description

Per-analyte mean, standard deviation, and coefficient of variation (sd / mean) of pixel intensities, by well-group type. Calibrator / standard wells are reported separately so calibration variability is easy to inspect.

Usage

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

Arguments

object	A qview object returned by read_qview().
...	Unused; for S3 generic compatibility.

Value

A tibble::tibble() with one row per well_type x analyte combination, columns: well_type, analyte, unit, n, mean, sd, cv, min, max.

See Also

Other qview-methods: as_tibble.qview(), is_qview(), plot.qview(), print.qview(), print.qview_summary()

Examples

```
path <- system.file("extdata", "example.Q-View", package = "qviewparsR")
qv <- read_qview(path, verbose = FALSE)
summary(qv)
```

---

well_label *Convert plate row / column to a well label*

Description

[Experimental]

Usage

```
well_label(row, col, zero_based = FALSE)
```

Arguments

row	Integer or character. Either a 0/1-based row index or already a single letter ("A"...).
col	Integer column index (1-based or 0-based; values >= 1 are treated as 1-based).
zero_based	Logical. If TRUE, treat row and col as 0-based; if FALSE (default), treat them as 1-based.

Details

Converts 0-based or 1-based row and column indices to the standard "A1" ... "H12" plate notation. Vectorised.

Value

Character vector of well labels (e.g. "A1", "H12"), recycled to the longer of row / col.

See Also

Other qview-helper: strip_qview_prefix()

Examples

```
well_label(0, 0, zero_based = TRUE)   # "A1"
well_label(7, 11, zero_based = TRUE)  # "H12"
well_label("C", 5)                    # "C5"
```

`write_qview` *Write a Q-View object to disk*

Description

[Experimental]

Usage

```
write_qview_xlsx(
  qv,
  path,
  template = NULL,
  overwrite = FALSE,
  call = rlang::caller_env()
)

write_qview_csv(qv, path, template = NULL, call = rlang::caller_env())

write_qview_rds(qv, path, overwrite = FALSE, call = rlang::caller_env())

qview_to_xlsx(
  qv,
  path,
  template = NULL,
  overwrite = FALSE,
  call = rlang::caller_env()
)

qview_to_csv_dir(qv, dir, template = NULL, call = rlang::caller_env())
```

Arguments

`qv`	A `qview` object from `read_qview()`.
`path`	Output path. For `write_qview_xlsx()` / `write_qview_rds()` this is a single file path; for `write_qview_csv()` it is the output directory (created if it does not exist).
`template`	Optional plate-template tibble (from `read_qview_template()`) to include as an extra sheet / file.

overwrite	Logical. If FALSE (the default), an existing destination triggers an error; set to TRUE to replace it. Ignored by write_qview_csv() (it only adds files).
call	The execution environment of the calling function. Used for error reporting; experts only.
dir	Deprecated alias for path accepted by qview_to_csv_dir(). Use path instead.

Details

Three writers, one for each common destination. All return the input qv invisibly so they compose with the pipe.

- write_qview_xlsx() – one sheet per parsed table.
- write_qview_csv() – one CSV per parsed table inside a directory.
- write_qview_rds() – a single .rds containing the full qview object (lossless, the only round-trippable format).

Value

qv, invisibly, to support pipelines.

Examples

```
path <- system.file("extdata", "example.Q-View", package = "qviewparsR")
qv <- read_qview(path, verbose = FALSE)

out <- tempfile()
dir.create(out)
qv |>
  write_qview_rds(file.path(out, "plate.rds")) |>
  write_qview_csv(file.path(out, "plate_csv"))
list.files(out, recursive = TRUE)
```

Index