Package: AquaSensR (via r-universe)

June 2, 2026

Title Quality Control and Analysis of Continuous Water Quality Data

Version 0.0.0.9000

Description Methods for quality control and exploratory analysis of continuous surface water quality data. Functions are developed to facilitate data formatting for the Water Quality Exchange Network <https://www.epa.gov/waterdata/water-quality-data-upload-wqx> and reporting of quality control to state agencies.

Depends R (>= 4.1.0)

Imports bslib, dataRetrieval, dplyr, DT, lubridate, plotly, readxl, shiny, tidyr, writexl, zip

License CC0

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

URL <https://github.com/massbays-tech/AquaSensR>, <https://massbays-tech.github.io/AquaSensR/>

BugReports https://github.com/massbays-tech/AquaSensR/issues

LazyData true

LazyDataCompression xz

Suggests covr, here, knitr, patchwork, quarto, testthat (>= 3.0.0), webshot2

Config/testthat/edition 3

VignetteBuilder quarto

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev make libicu-dev libuv1-dev libxml2-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev libx11-dev zlib1g-dev

Repository https://massbays-tech.r-universe.dev

Date/Publication 2026-06-02 17:17:56 UTC

RemoteUrl https://github.com/massbays-tech/AquaSensR

RemoteRef HEAD

RemoteSha 6e443922309ef7a5bd0d36b65acd16769d58ce32

Contents

`anlzASRflag` *Plot QC flag results for a continuous monitoring parameter*

Description

Plot QC flag results for a continuous monitoring parameter

Usage

```
anlzASRflag(flag, overlay = NULL)
```

Arguments

`flag`	data frame returned by `utilASRflag`.
`overlay`	optional two-column data frame (e.g., `contdat`) with a `DateTime` column and one numeric parameter column. When supplied, the series is drawn as a light blue line on a second y-axis on the right side of the plot. Pass `NULL` (the default) to omit the overlay.

Details

Produces an interactive **plotly** time series showing all observations as a line, with non-passing observations overlaid as markers. Marker **colour** indicates which QC check fired:

- **Gross range** — red
- **Spike** — orange
- **Rate of change** — purple
- **Flatline** — blue

Marker **shape** indicates severity:

- **Suspect** — upward triangle
- **Fail** — cross (×)

An observation flagged by multiple checks appears as a marker for each check that fired, allowing all sources of concern to be visible.

Value

An interactive `plotly` object.

See Also

`editASRflag`, which uses this function internally to render the plot inside a Shiny app.

Examples

```
contpth <- system.file('extdata/ExampleCont1.xlsx', package = 'AquaSensR')
dqopth <- system.file('extdata/ExampleDQO.xlsx', package = 'AquaSensR')

contdat <- readASRcont(contpth, runchk = FALSE)
dqodat <- readASRdqo(dqopth, runchk = FALSE)

flagdat <- utilASRflag(contdat, dqodat, param = 'Water_Temp_C')
anlzASRflag(flagdat)
```

---

checkASRcont *Check continuous monitoring data*

Description

Check continuous monitoring data

Usage

```
checkASRcont(contdat)
```

Arguments

contdat	input data frame for results

Details

This function is used internally within readASRcont to run several checks on the input data to verify correct formatting before downstream analysis.

The input data can use either of two formats:

- **Separate columns**: Date, Time, and at least one parameter column
- **Combined column**: DateTime, and at least one parameter column

The following checks are made:

- Column names: Should include only Date, Time, DateTime, and at least one parameter column that matches the Parameter column in paramsASR
- Required columns are present: Either Date + Time or DateTime are required for downstream analysis and upload to WQX
- At least one parameter column is present: At least one parameter column that matches the Parameter column in paramsASR is required for downstream analysis and upload to WQX
- Date format (separate columns only): Should be parseable by lubridate::parse_date_time() using year-first ("2024-06-01"), month-first ("06/01/2024"), or day-first ("01/06/2024") formats
- Time format (separate columns only): Should be parseable by lubridate::parse_date_time() using 24-hour ("16:30:33"), 12-hour AM/PM ("4:30:33 PM"), or Excel-prefixed ("1899-12-31 16:30:33") formats
- DateTime format (combined column only): Should be parseable by lubridate::parse_date_time() using year-first, month-first, or day-first date order combined with 24-hour or 12-hour AM/PM time (e.g. "2024-06-01 16:30:33", "06/01/2024 16:30:33", or "2024-06-01 4:30:33 PM")
- Missing values: Missing values in parameter columns produce a warning rather than an error, since cleaned data files may legitimately contain NA values. Missing values in DateTime, Date, or Time columns still cause an error.
- Parameter columns should be numeric: All parameter columns should be numeric values

Value

contdat is returned as is if no errors are found. An informative error is raised for structural problems (unrecognised column names, missing required columns, unparseable date/time values, or non-numeric parameter values). Missing values in parameter columns produce a warning instead of an error.

Examples

```
contpth <- system.file('extdata/ExampleCont1.xlsx', package = 'AquaSensR')

contdat <- utilASRimportcont(contpth)

checkASRcont(contdat)
```

checkASRdqo *Check data quality objectives*

Description

Check data quality objectives

Usage

```
checkASRdqo(dqodat)
```

Arguments

dqodat	input data frame of data quality objectives

Details

This function is used internally within readASRdqo to run several checks on the input data to verify correct formatting before downstream analysis.

The following checks are made:

- Column names: Should include only Parameter, Flag, GrMin, GrMax, Spike, FlatN, FlatDelta, RoCStDv, and RoCHours
- All columns present: All columns from the previous check should be present
- At least one parameter is present: At least one parameter in the Parameter column matches the Parameter column in paramsASR
- Parameter format: All parameters listed in the Parameter column should match those in the Parameter column in paramsASR
- Flag column: The Flag column should contain only "Fail" or "Suspect" entries
- Numeric columns: All columns except Parameter and Flag should be numeric values

Value

dqodat is returned as is if no errors are found, otherwise an informative error message is returned prompting the user to make the required correction to the raw data before proceeding.

Examples

```
library(dplyr)

dqopth <- system.file('extdata/ExampleDQO.xlsx', package = 'AquaSensR')

dqodat <- suppressWarnings(readxl::read_excel(dqopth, na = c('NA', 'na', ''),
     guess_max = Inf))

checkASRdqo(dqodat)
```

`editASRdrift` *Interactive drift correction editor*

Description

Opens a Shiny application for interactively correcting instrument drift in continuous water quality monitoring data. Click the plot twice to mark the start and end of a drift period, enter the reference value measured by an independent calibrated instrument at the end of the deployment, and click **Apply Correction**. A third click resets the selection.

Usage

```
editASRdrift(cont)
```

Arguments

`cont`	`contdat` data frame returned by `readASRcont`

Details

How to correct drift: Zoom and pan with the plot toolbar to identify the drift period. Click once to set the start time and click again to set the end time. Clicking a third time resets the selection. Once two times are selected, enter the **Reference value** (the true reading from an independent calibrated instrument at the end of the deployment) and click **Apply Correction**.

The `cal_check` value (the deployed sensor reading at the end of the window) is inferred automatically from the data. The correction is distributed linearly across the window: zero at the start, full correction at the end. See `utilASRdrift` for the algorithm.

After a correction is applied, the plot retains the original (pre-correction) values for the window as a solid gray line so the adjustment can be assessed visually. A red circle marks the supplied reference value at the end of the window. These elements are display-only and are not included in the returned data.

Multiple corrections can be applied per parameter (e.g., one per deployment period), and each can be individually undone.

Controls:

- **Parameter**: drop-down selector to switch between parameters. Corrections are tracked independently for each parameter.
- **Undo Last Correction**: reverses the most recently applied correction for the current parameter.
- **Start Over**: restores all original values for every parameter and clears the corrections log.
- **Export Progress**: saves the current corrected data and corrections log as Excel files in a ZIP archive.
- **Done / Close**: stops the app and returns the corrected data and corrections summary to the R session.

Value

A list with two elements, invisibly returned after the app closes:

`contdat` A data frame with the same structure as the input `cont` (sorted by `DateTime`), with drift-corrected values replacing the originals in all corrected windows.

`corrections` A data frame summarising every correction applied, with columns `Parameter`, `drift_start`, `drift_end`, `cal_ref`, `cal_check`, and `drift_applied`.

Examples

```
## Not run:
contpth <- system.file("extdata/ExampleCont1.xlsx", package = "AquaSensR")
contdat <- readASRcont(contpth)
result  <- editASRdrift(contdat)

## End(Not run)
```

---

editASRflag *Interactive editor for continuous monitoring data*

Description

Opens a Shiny application displaying the QC flag plot from `anlzASRflag` for each parameter in `contdat` and allows the user to interactively select and remove data points. Points are removed by clicking or drawing a selection using the box or lasso tool on the plot. A running table of removed points (including their flag assignments) is shown in the sidebar and is specific to the currently displayed parameter. Individual removal batches can be undone, or all parameters can be fully reset. Clicking **Done / Close** stops the app and returns the filtered datasets for all parameters to the R session.

Usage

```
editASRflag(cont, dqo)
```

Arguments

`cont`	`contdat` data frame returned by `readASRcont`
`dqo`	`dqodat` data frame returned by `readASRdqo`

Details

QC flags are computed internally via `utilASRflagall`.

How to select points: Zooming and panning with the plot toolbar is recommended to more easily identify points for removal. These options are available in the menu on the top right when hovering over the plot.

Points can be selected for removal three ways. First, individual points can be removed by clicking. Second and third, use the box or lasso selection tool by hovering over the plot and selecting the desired tool from the menu on the top right. Click and drag over the desired area for the box selection or click and encircle the points with the lasso tool to add the points to the removal table. Double-click the plot background to remove the selected area if present after removal.

Controls:

- **Parameter**: drop-down selector to switch between parameters. Edits to each parameter are preserved independently when switching.
- **Overlay**: optional drop-down to display a second parameter from `condtat` on a right-side y-axis, useful for spotting co-occurring changes across parameters.
- **USGS Overlay**: enter a USGS site number and select a parameter type, then click **Load** to fetch continuous data from NWIS and display it on the secondary y-axis. Loading USGS data clears any contdat overlay and selecting a contdat overlay clears the USGS data. Site numbers can be found at the NWIS Mapper (`https://apps.usgs.gov/nwismapper`).
- **Linked Removal**: optional checkbox. When checked, any timestamps removed from the current parameter are simultaneously removed from all other parameters. Undo restores the current parameter and all other parameters together as a single operation, regardless of which parameter is active when undo is clicked.
- **Undo Last Removal**: restores the most recently removed point or batch of points. If the removal was linked, all affected parameters are restored together.
- **Start Over**: restores all removed points for every parameter and resets all DQO thresholds to their original values.
- **Export Progress**: saves the current cleaned data and DQO thresholds as Excel files in a ZIP archive. If any points have been removed, a removed-observations file is included as well.
- **Done / Close**: stops the app and returns the filtered datasets for all parameters to the R session.

DQO Settings panel: A collapsible panel on the right side of the plot exposes the numeric QC thresholds for the currently selected parameter. Each of the four checks (gross range, spike, rate of change, flatline) shows independent **Suspect** and **Fail** threshold columns.

- **Apply**: re-computes flags for the current parameter using the edited thresholds. Previously removed points are retained.
- **Reset to original**: reverts the inputs to the values supplied in `dqo` and re-computes flags. Any points already removed are retained.

Threshold edits are per-parameter and independent; switching parameters shows that parameter's current thresholds without affecting others.

The app is constructed inline so that flag data are available directly to the server without file I/O. `shiny::runApp()` blocks until `shiny::stopApp()` is called by the Done button; its return value becomes the function return value.

Value

A list with three elements, invisibly returned after the app closes:

`contdat` A data frame with the same structure as the input `contdat` (sorted by `DateTime`), where values removed by the user are replaced with `NA`. Rows in which every parameter was removed are retained with only `DateTime` populated.

`dqodat` A data frame with the same structure as the input `dqo`, reflecting any threshold edits made in the DQO Settings panel. If no edits were made the values are identical to the input.

`removed` A data frame of all removed observations across all parameters, with columns `Parameter`, `DateTime`, `gross_flag`, `spike_flag`, `roc_flag`, and `flat_flag`.

Examples

```
## Not run:
contpth <- system.file("extdata/ExampleCont1.xlsx", package = "AquaSensR")
dqopth  <- system.file("extdata/ExampleDQO.xlsx", package = "AquaSensR")
contdat <- readASRcont(contpth)
dqodat  <- readASRdqo(dqopth)
cleaned <- editASRflag(contdat, dqodat)

## End(Not run)
```

`formASRcont` *Format continuous data*

Description

Format continuous data

Usage

```
formASRcont(contdat, tz = "Etc/GMT+5")
```

Arguments

`contdat`	input data frame
`tz`	character string of time zone for the date and time columns, defaults to Etc/GMT+5 (Eastern time zone, no daylight savings). See `OlsonNames()` for acceptable time zones.

Details

This function is used internally within `readASRcont` to format the input data for downstream analysis. The formatting includes:

- Combine Date and Time columns (separate column format only): The `Time` column is parsed flexibly using `lubridate::parse_date_time()` (accepting 24-hour, 12-hour AM/PM, and Excel-prefixed formats) and reformatted to `HH:MM:SS` before being united with `Date` into a single `DateTime` column, which is then converted to POSIXct using `parse_date_time()` with year-first, month-first, and day-first date orders.
- Convert DateTime to POSIXct (combined column format only): The `DateTime` column is parsed flexibly using `lubridate::parse_date_time()` with year-first, month-first, and day-first date orders combined with 24-hour and 12-hour AM/PM time formats, and converted to POSIXct with the specified time zone.
- Convert non-numeric columns to numeric: Converts all columns except DateTime to numeric if they are not already.

Value

A formatted data frame of the continuous data

Examples

```
contpth <- system.file('extdata/ExampleCont1.xlsx', package = 'AquaSensR')

contdat <- utilASRimportcont(contpth)

formASRcont(contdat)
```

`formASRdqo` *Format data quality objectives*

Description

Format data quality objectives

Usage

```
formASRdqo(dqodat)
```

Arguments

`dqodat`	input data frame

Details

This function is used internally within `readASRdqo` to format the input data for downstream analysis. The formatting includes:

- Convert non-numeric columns to numeric: Converts all columns except `Parameter` and `Flag` to numeric if they are not already.

Value

A formatted data frame of the data quality objectives

Examples

```
dqopth <- system.file('extdata/ExampleDQO.xlsx', package = 'AquaSensR')

dqodat <- suppressWarnings(readxl::read_excel(dqopth, na = c('NA', 'na', ''),
     guess_max = Inf))

formASRdqo(dqodat)
```

---

paramsASR *Master list and units for acceptable parameters*

Description

Master list and units for acceptable parameters

Usage

```
paramsASR
```

Format

A `data.frame`

Details

This information is used to verify the correct format of input data and for formatting output data for upload to WQX. A column showing the corresponding WQX names is also included.

Examples

```
paramsASR
```

readASRcont *Read continuous monitoring data from an external file*

Description

Read continuous monitoring data from an external file

Usage

```
readASRcont(contpth, tz = "Etc/GMT+5", runchk = TRUE)
```

Arguments

contpth	character string of path to the continuous data file. Supported formats are Excel (.xlsx), CSV (.csv), or comma-delimited text (.txt).
tz	character string of time zone for the date and time columns, defaults to Etc/GMT+5 (Eastern time zone, no daylight savings). See OlsonNames() for acceptable time zones.
runchk	logical to run data checks with checkASRcont

Details

For Excel files the file is imported via utilASRimportcont, which forces Date, Time, and DateTime columns to character and converts Excel numeric serial representations to human-readable strings. Excel files must not be open in another program (e.g. Excel, LibreOffice) when this function is run.

For CSV and comma-delimited text files the file is read with read.csv, with Date, Time, and DateTime columns forced to character and all other columns type-guessed. No lock-file check is performed for these formats.

Always verify the correct time zone for your data. If your data are in a different time zone than Etc/GMT+5 (default), specify the correct time zone in the tz argument.

Value

A formatted continuous monitoring data frame that can be used for downstream analysis

Examples

```
contpth <- system.file('extdata/ExampleCont2.xlsx', package = 'AquaSensR')
readASRcont(contpth)

contpth <- system.file('extdata/ExampleCont1.csv', package = 'AquaSensR')
readASRcont(contpth)

contpth <- system.file('extdata/ExampleCont2.txt', package = 'AquaSensR')
readASRcont(contpth)
```

readASRdqo *Read data quality objectives from an external file*

Description

Read data quality objectives from an external file

Usage

```
readASRdqo(dqopth, runchk = TRUE)
```

Arguments

dqopth	character string of path to the data quality objectives file
runchk	logical to run data checks with checkASRdqo

Details

The file must not be open in another program (e.g. Excel, LibreOffice) when this function is run, otherwise an error will indicate to close the file before proceeding.

Value

A formatted data quality objectives data frame that can be used for downstream analysis

Examples

```
dqopth <- system.file('extdata/ExampleDQO.xlsx', package = 'AquaSensR')

readASRdqo(dqopth)
```

readASRusgs *Retrieve USGS time series data for overlay in editASRflag*

Description

Downloads unit-value (continuous) data from the USGS Water Data API for a given site and parameter over a specified date range. The result is a two-column data frame compatible with the overlay argument of anlzASRflag and the USGS Overlay feature in editASRflag.

Usage

```
readASRusgs(site, pcode, start, end, tz = "Etc/GMT+5")
```

Arguments

site	Character. USGS site number (typically 8 digits, e.g. "01099500"). Find site numbers using the NWIS Mapper at https://apps.usgs.gov/nwismapper.
pcode	Character. Five-digit USGS parameter code. Common codes: "00060" Discharge / streamflow (ft^3/s) "00065" Gage height (ft) "00045" Precipitation (in)
start, end	Date range as Date objects or "YYYY-MM-DD" character strings.
tz	Character. Time zone to which the returned DateTime column is converted. Defaults to 'Etc/GMT+5' (Eastern Standard Time, no daylight saving), matching the default in readASRcont and formASRcont. Pass the same value you used in readASRcont() so the overlay aligns correctly with the primary time series in anlzASRflag and editASRflag. See OlsonNames() for valid strings.

Details

Data are fetched via dataRetrieval::read_waterdata_continuous(), which targets the modern USGS Water Data API (https://api.waterdata.usgs.gov). The API returns timestamps in UTC and readASRusgs() re-expresses them in tz via lubridate::with_tz() so the result aligns with contdat without shifting the underlying moments in time.

The station name shown in the editASRflag status line is retrieved with a second lightweight call to dataRetrieval::read_waterdata_monitoring_location(). If that call fails the site number is used as a fallback.

An error is raised if the site does not record the requested parameter, if the date range returns no observations, or if the API is unreachable.

Value

A two-column data frame with columns DateTime (POSIXct, in the timezone given by tz) and a second column whose name is a human-readable label combining the parameter description and site number (e.g. "Streamflow (ft\u00b3/s) [01099500]"). The data frame carries a "site_name" attribute containing the station name, used by editASRflag for the status message.

Examples

```
## Not run:
# Fetch streamflow for the Concord R Below R Meadow Brook, at Lowell, MA
# 2024-01-01 to 2024-01-02
flow <- readASRusgs("01099500", "00060", "2024-01-01", "2024-01-02")
head(flow)

## End(Not run)
```

`utilASRdrift` *Apply linear drift correction to a continuous monitoring parameter*

Description

Corrects for instrument drift over a specified window using a linear interpolation approach. The correction at the start of the window is zero and grows linearly to `cal_ref - cal_check` at the end, where `cal_check` is inferred from the data as the sensor reading at `drift_end_time`.

Usage

```
utilASRdrift(
  cont,
  param,
  cal_ref,
  drift_start_time,
  drift_end_time,
  plot = FALSE
)
```

Arguments

`cont`	`contdat` data frame returned by `readASRcont`
`param`	character string naming the parameter column to correct
`cal_ref`	numeric; the true or accepted value measured by an independent calibrated instrument at the end of the deployment period
`drift_start_time`	start of the drift window (POSIXct or coercible)
`drift_end_time`	end of the drift window (POSIXct or coercible)
`plot`	logical; `FALSE` (default) the corrected data frame is returned. If `TRUE` a plotly object is returned instead, showing the corrected time series (blue), the original values within the drift window (gray), and the reference value at the end of the window (red circle).

Details

The `cal_check` value (what the deployed sensor was actually reading at `drift_end_time`) is inferred directly from the data, so only the independent reference reading (`cal_ref`) needs to be supplied. The total drift `cal_ref - cal_check` is distributed linearly across the window: zero correction is applied at `drift_start_time` and the full correction is applied at `drift_end_time`.

This correction formula is as follows:

$$\text{adj} = \text{sensor} + (\text{cal\_ref} - \text{cal\_check}) \times \frac{i - i_{\min}}{i_{\max} - i_{\min}}$$

Value

If `plot = FALSE`, a copy of `cont` with corrected values for `param` in the drift window (values outside the window are unchanged). If `plot = TRUE`, a `plotly` object.

Examples

```
contpth <- system.file("extdata/ExampleCont1.xlsx", package = "AquaSensR")
contdat <- readASRcont(contpth, runchk = FALSE)
t1 <- min(contdat$DateTime)
t2 <- max(contdat$DateTime)
utilASRdrift(contdat, "Water_Temp_C", cal_ref = 26, t1, t2)
```

`utilASRflag` *Flag continuous monitoring data with QC criteria*

Description

Flag continuous monitoring data with QC criteria

Usage

```
utilASRflag(cont, dqo, param)
```

Arguments

`cont`	`condtat` data frame returned by `readASRcont`
`dqo`	`dqodat` data frame returned by `readASRdqo`
`param`	character string naming the parameter column to evaluate. Must match one of the parameter columns present in `contdat`. If `param` has no matching entry in `dqodat$Parameter` a warning is returned and all flags are returned as `"pass"`.

Details

Applies four independent QC checks to the selected parameter in `contdat`, matching thresholds from `dqodat` by `Parameter`. Each check produces its own flag (`"pass"`, `"suspect"`, or `"fail"`) so the user can see exactly which criteria fired. Thresholds are read from the two rows in `dqodat` that match the parameter — one with `Flag == "Fail"` and one with `Flag == "Suspect"`.

Gross range (`gross_flag`) — Observations below `GrMin` or above `GrMax` in the `"Fail"` row are flagged `"fail"`. Observations below `GrMin` or above `GrMax` in the `"Suspect"` row (but within the fail bounds) are flagged `"suspect"`.

Spike (`spike_flag`) — The absolute difference between consecutive observations is compared to `Spike` in the `"Fail"` row (fail) and `Spike` in the `"Suspect"` row (suspect). The second observation in the jump is flagged.

Rate of change (`roc_flag`) — For each observation the standard deviation of all raw values within a trailing `RoCHours`-hour window is multiplied by `RoCStDv` to produce a threshold. The observation is flagged `"suspect"` if its absolute lag-1 difference exceeds that threshold using the `"Suspect"`

row thresholds, and "fail" using the "Fail" row thresholds. Each row is checked independently; if RoCStDv or RoCHours is NA for a row that severity level is skipped. Requires at least 2 values in the window; otherwise "pass".

Flatline (flat_flag) — Observations accumulate run length as long as the range (max minus min) of all values in the current run is strictly less than FlatDelta. Observations whose run length reaches FlatN are flagged, using the "Suspect" row thresholds for suspect and the "Fail" row thresholds for fail.

Any threshold value set to NA in dqodat is silently skipped. The corresponding severity level is not applied and affected observations remain "pass" for that check. This applies to both the "Suspect" and "Fail" rows independently, so individual checks or severity levels can be disabled selectively.

Data are sorted by DateTime before processing.

Underlying concepts and code for this function borrow heavily from those in the ContDataQC package. Any credit for the approach should go to the ContDataQC authors.

Value

A data frame with columns DateTime, the selected parameter, and four flag columns: gross_flag, spike_flag, roc_flag, and flat_flag.

Examples

```
contpth <- system.file('extdata/ExampleCont1.xlsx', package = 'AquaSensR')
dqopth <- system.file('extdata/ExampleDQO.xlsx', package = 'AquaSensR')

contdat <- readASRcont(contpth, runchk = FALSE)
dqodat <- readASRdqo(dqopth, runchk = FALSE)

utilASRflag(cont = contdat, dqo = dqodat, param = 'Water_Temp_C')
```

---

utilASRflagall *Apply QC flags to all parameters in continuous monitoring data*

Description

A wrapper around utilASRflag that iterates over every parameter in contdat the results as a named list.

Usage

```
utilASRflagall(contdat, dqodat)
```

Arguments

contdat	data frame returned by readASRcont
dqodat	data frame returned by readASRdqo

Details

Parameters are defined as every column in `contdat` other than `DateTime`. If a parameter has no matching entry in `dqodat$Parameter` all four of its flags are returned as `"pass"`.

Each element of the returned list is the data frame produced by `utilASRflag` for that parameter: columns `DateTime`, the parameter, `gross_flag`, `spike_flag`, `roc_flag`, and `flat_flag`.

Value

A named list of data frames, one per matched parameter, with names equal to the parameter column names.

Examples

```
contpth <- system.file('extdata/ExampleCont1.xlsx', package = 'AquaSensR')
dqopth <- system.file('extdata/ExampleDQO.xlsx', package = 'AquaSensR')

contdat <- readASRcont(contpth, runchk = FALSE)
dqodat <- readASRdqo(dqopth, runchk = FALSE)

utilASRflagall(contdat, dqodat)
```

---

utilASRflagflatline *Apply flatline QC flag*

Description

Apply flatline QC flag

Usage

```
utilASRflagflatline(flag, vals, dqo)
```

Arguments

`flag`	character vector of current flag values (`"pass"`, `"suspect"`, or `"fail"`).
`vals`	numeric vector of observed values, the same length as `flag`.
`dqo`	two-row data frame of data quality objectives for the parameter being checked, containing one row where `Flag == "Fail"` and one where `Flag == "Suspect"`. Optional numeric columns `FlatN` and `FlatDelta` define the run-length and tolerance thresholds for each severity level. Either row may have `NA` for these columns, in which case that level of check is skipped.

Details

Uses `utilASRflagrleflat` to compute consecutive run lengths. A run extends as long as the range (max minus min) of all values in the run so far is strictly less than `FlatDelta`. An observation is flagged `"suspect"` when its run length reaches `FlatN` (using `FlatDelta` from the `"Suspect"` row), and `"fail"` when its run length reaches `FlatN` (using `FlatDelta` from the `"Fail"` row).

Value

Updated character flag vector.

Examples

```
flag <- rep("pass", 8)
vals <- c(10, 10, 10.005, 10.002, 10.001, 10.003, 12, 12)
dqo <- data.frame(
  Flag = c("Fail", "Suspect"),
  FlatN = c(5, 3), FlatDelta = c(0.01, 0.01)
)
utilASRflagflatline(flag, vals, dqo)
```

---

utilASRflaggross *Apply gross range QC flag*

Description

Apply gross range QC flag

Usage

```
utilASRflaggross(flag, vals, dqo)
```

Arguments

flag	character vector of current flag values ("pass", "suspect", or "fail").
vals	numeric vector of observed values, the same length as flag.
dqo	two-row data frame from data quality objectives for the parameter being checked, containing one row where Flag == "Fail" and one where Flag == "Suspect". Must contain numeric columns GrMin and GrMax.

Details

Observations below GrMin or above GrMax in the "Fail" row are flagged "fail". Observations below GrMin or above GrMax in the "Suspect" row (but within the fail bounds) are flagged "suspect". NA threshold values are silently skipped.

Value

Updated character flag vector.

Examples

```
flag <- rep("pass", 5)
vals <- c(-2, 0, 15, 26, 32)
dqo <- data.frame(
  Flag = c("Fail", "Suspect"),
  GrMin = c(-1, 0), GrMax = c(30, 25)
)
utilASRflaggross(flag, vals, dqo)
```

`utilASRflagrleflat` *Compute consecutive run lengths for flatline detection*

Description

Compute consecutive run lengths for flatline detection

Usage

```
utilASRflagrleflat(vals, delta)
```

Arguments

`vals`	numeric vector of observed values.
`delta`	non-negative numeric scalar tolerance. An observation extends the current run only when the range (max minus min) of all values in the run so far, including the new observation, is strictly $<$ `delta`. If this condition fails the run resets.

Details

For each position i, the run extends only when adding the current observation to the run keeps the range (max minus min of all values in the run) strictly $<$ `delta`. This prevents both large single-step jumps and slow cumulative drift from accumulating run length. A range equal to `delta` is not considered flatline. A run length of 1 means the observation is not part of a flat stretch. `NA` values in `vals` break the run.

Value

Integer vector the same length as `vals` giving the run length at each position.

Examples

```
vals <- c(10, 10, 10.005, 10.003, 12, 12, 12)
utilASRflagrleflat(vals, delta = 0.01)
```

`utilASRflagroc` *Apply rate-of-change QC flag*

Description

Apply rate-of-change QC flag

Usage

```
utilASRflagroc(flag, vals, datetimes, dqo)
```

Arguments

`flag`	character vector of current flag values (`"pass"`, `"suspect"`, or `"fail"`).
`vals`	numeric vector of observed values, the same length as `flag`.
`datetimes`	POSIXct vector of observation timestamps, the same length as `flag`.
`dqo`	two-row data frame from data quality objectives for the parameter being checked, containing one row where `Flag == "Fail"` and one where `Flag == "Suspect"`. Each row's numeric columns `RoCStDv` (SD multiplier) and `RoCHours` (trailing window width in hours) control the check independently. If either column is `NA` for a given row that severity level is skipped entirely.

Details

For each observation the standard deviation of all raw values within a trailing `RoCHours`-hour window ending just before (and excluding) that observation is multiplied by `RoCStDv` to produce a threshold. The observation is flagged if the absolute lag-1 difference exceeds that threshold — `"suspect"` using the `"Suspect"` row thresholds and `"fail"` using the `"Fail"` row thresholds. At least 2 values must fall within the window to compute the standard deviation; otherwise the observation is skipped. Flags are only ever upgraded (pass -> suspect -> fail), never downgraded.

Value

Updated character flag vector.

Examples

```
flag <- rep("pass", 6)
vals <- c(10, 10.2, 10.1, 10.3, 15.0, 10.2)
datetimes <- as.POSIXct("2024-01-01") + seq(0, 5) * 900  # 15-min intervals
dqo <- data.frame(Flag = c("Fail", "Suspect"), RoCStDv = c(2, 3), RoCHours = c(2, 2))
utilASRflagroc(flag, vals, datetimes, dqo)
```

`utilASRflagspike` *Apply spike QC flag*

Description

Apply spike QC flag

Usage

```
utilASRflagspike(flag, vals, dqo)
```

Arguments

`flag`	character vector of current flag values (`"pass"`, `"suspect"`, or `"fail"`).
`vals`	numeric vector of observed values, the same length as `flag`.
`dqo`	two-row data frame from data quality objectives for the parameter being checked, containing one row where `Flag == "Fail"` and one where `Flag == "Suspect"`. Optional numeric column `Spike` defines the absolute-difference threshold for each severity level. Either row may have `NA` for `Spike`, in which case that level of check is skipped.

Details

The absolute difference between each observation and the preceding one is computed. If the difference is greater than or equal to `Spike` in the `"Suspect"` row the observation is flagged `"suspect"`; greater than or equal to `Spike` in the `"Fail"` row flags `"fail"`. The first observation always receives `NA` for the difference and is not flagged by this check.

Value

Updated character flag vector.

Examples

```
flag <- rep("pass", 5)
vals <- c(10, 10.5, 14, 10.2, 10.3)
dqo <- data.frame(Flag = c("Fail", "Suspect"), Spike = c(2.0, 1.5))
utilASRflagspike(flag, vals, dqo)
```

utilASRflagupdate *Update QC flag severity*

Description

Update QC flag severity

Usage

```
utilASRflagupdate(flag, level, condition)
```

Arguments

flag	character vector of current flag values; each element must be one of "pass", "suspect", or "fail".
level	scalar character string — the new flag level to apply ("pass", "suspect", or "fail").
condition	logical vector the same length as flag. Elements that are TRUE and whose current flag is less severe than level will be upgraded. NA values in condition are treated as FALSE.

Details

Severity is ordered "pass" < "suspect" < "fail". A flag is only ever upgraded, never downgraded.

Value

Character vector the same length as flag with flags updated where condition is TRUE and level is more severe than the existing flag.

Examples

```
flag <- c("pass", "pass", "suspect", "fail")
utilASRflagupdate(flag, "suspect", c(TRUE, FALSE, TRUE, TRUE))
utilASRflagupdate(flag, "fail",    c(TRUE, TRUE, FALSE, FALSE))
```

utilASRimportcont *Import continuous monitoring data from an Excel file*

Description

Import continuous monitoring data from an Excel file

Usage

```
utilASRimportcont(contpth)
```

Arguments

contpth	character string of path to the continuous data file

Details

Reads an Excel workbook and returns a data frame with Date, Time, and DateTime columns preserved as character strings, with Excel numeric representations converted to human-readable text:

- **Date**: integer-like strings (Excel date serial numbers, e.g. "45518") are converted to yyyy-mm-dd using Excel's origin of 1899-12-30.
- **Time**: decimal fraction strings between 0 and 1 (Excel time fractions, e.g. "0.58105") are converted to HH:MM:SS.
- **DateTime**: numeric strings with an integer part (Excel datetime serials, e.g. "45518.58105") are converted to yyyy-mm-dd HH:MM:SS.
- Text values in any of these columns (e.g. "2024-08-14", "4:30:33 PM") are left unchanged.

This function is called internally by readASRcont and can also be used to prepare data for manual use with checkASRcont or formASRcont.

Value

A data frame with date/time columns as character strings and all other columns type-guessed by readxl.

Examples

```
contpth <- system.file('extdata/ExampleCont1.xlsx', package = 'AquaSensR')

utilASRimportcont(contpth)
```

---

utilASRopencheck *Check if an Excel file is open and execute a read function*

Description

Check if an Excel file is open and execute a read function

Usage

```
utilASRopencheck(pth, fn)
```

Arguments

pth	character string path to the Excel file
fn	a zero-argument function that reads the file

Details

First checks for lock files created by Excel (`~$filename`) and LibreOffice (`.~lock.filename#`). If none are found, calls `fn()` and catches the `utils::unzip` error that occurs when Excel holds an OS-level lock without creating a local lock file (e.g. on OneDrive). Both paths produce the same user-facing message.

Value

The value returned by `fn()`.

Examples

```
contpth <- system.file('extdata/ExampleCont1.xlsx', package = 'AquaSensR')
utilASRopencheck(contpth, \() readxl::read_excel(contpth, n_max = 0))
```

Index