Package 'isoreader2'

November 9, 2026

Title Read Stable Isotope Data Files

Version 0.7.1

Description Interface to the raw data and metadata stored in the file formats commonly encountered in scientific disciplines that make use of stable isotopes. Supports Isodat (.dxf, .cf, .did, .caf, .scn), IonOS (.iarc), LyticOS (.larc), Callisto (.bch), and Qtegra (.imexp) file formats. Provides a consistent data structure together with tools to aggregate, convert signal units, filter, and visualize the extracted data. The approach is described in Kopf et al. (2021) <doi:10.21105/joss.02878>.

License AGPL (>= 3)

Encoding UTF-8

URL https://isoreader2.isoverse.org/, https://github.com/isoverse/isoreader2

BugReports https://github.com/isoverse/isoreader2/issues

Depends R (>= 4.5.0)

Imports ggplot2, tools, utils, methods, stats, rlang (>= 1.1.0), cli (>= 3.6.0), lifecycle, processx, readr, tibble, dplyr, tidyr, tidyselect, purrr, withr, RcppSimdJson, scales, fansi, knitr

Suggests testthat (>= 3.0.0), vdiffr, arrow, openxlsx, rmarkdown

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Sebastian Kopf [aut, cre] (ORCID: <https://orcid.org/0000-0002-2044-0201>)

Maintainer Sebastian Kopf <sebastian.kopf@colorado.edu>

Repository CRAN

Date/Publication 2026-11-09 00:10:02 UTC

Contents

isoreader2-package . . . 2
auto . . . 3
c.ir_aggregated_data . . . 4
c.ir_isofiles . . . 4
ir_aggregate_isofiles . . . 5
ir_calculate_ratios . . . 6
ir_check_isoextract . . . 7
ir_convert_intensity . . . 9
ir_default_theme . . . 10
ir_examples_folder . . . 10
ir_export_to_excel . . . 11
ir_extract_isofiles . . . 12
ir_filter_for . . . 14
ir_filter_masses . . . 15
ir_find_isofiles . . . 16
ir_generate_tibble . . . 17
ir_get_data . . . 19
ir_get_problems . . . 21
ir_get_supported_file_types . . . 21
ir_isofiles_storage . . . 22
ir_metadata . . . 23
ir_options . . . 24
ir_plot_dual_inlet . . . 25
ir_plot_scans . . . 28
ir_plot_traces . . . 30
ir_read_isofiles . . . 33
ir_start_aggregator . . . 34
ir_storage . . . 36

Index **38**

`isoreader2-package` *isoreader2: Read Stable Isotope Data Files*

Description

Interface to the raw data and metadata stored in the file formats commonly encountered in scientific disciplines that make use of stable isotopes. Supports Isodat (.dxf, .cf, .did, .caf, .scn), IonOS (.iarc), LyticOS (.larc), Callisto (.bch), and Qtegra (.imexp) file formats. Provides a consistent data structure together with tools to aggregate, convert signal units, filter, and visualize the extracted data. The approach is described in Kopf et al. (2021) doi:10.21105/joss.02878.

Details

Resources:

- Website for the isoreader2 package: https://isoreader2.isoverse.org
- Package options: ir_options

Author(s)

Maintainer: Sebastian Kopf <sebastian.kopf@colorado.edu> (ORCID)

Authors:

- Sebastian Kopf <sebastian.kopf@colorado.edu> (ORCID)

See Also

Useful links:

- https://isoreader2.isoverse.org/
- https://github.com/isoverse/isoreader2
- Report bugs at https://github.com/isoverse/isoreader2/issues

---

auto *Automatic / default behavior*

Description

A sentinel that requests automatic behavior for an argument (currently the `data_type_as_facet` argument of the plotting functions `ir_plot_traces()`, `ir_plot_dual_inlet()`, `ir_plot_scans()`). It is the default for those arguments; pass `TRUE`/`FALSE` to override the automatic choice.

Usage

```
auto()
```

Value

an opaque sentinel of class `ir_auto`

c.ir_aggregated_data *Combine aggregated isofile data*

Description

Combine multiple `ir_aggregate_isofiles()` results into a single `ir_aggregated_data` object by row-binding each of the contained datasets (`metadata`, `traces`, `cycles`, `scans`, `resistors`, `vendor_data_table`, `problems`, ...) with `dplyr::bind_rows()`. Datasets present in only some of the objects are combined as well (missing columns are filled with `NA`). The file index `uidx` is re-numbered across the inputs so each file stays uniquely identified (and its datasets stay correctly linked) in the combined object.

Usage

```
## S3 method for class 'ir_aggregated_data'
c(...)
```

Arguments

`...`	`ir_aggregated_data` objects to combine

Value

a single combined `ir_aggregated_data` object

c.ir_isofiles *Combine isofiles*

Description

Combine multiple collections of isofiles (read by `ir_read_isofiles()`) into a single `ir_isofiles` object by row-binding them with `dplyr::bind_rows()`. This preserves the object structure and type.

Usage

```
## S3 method for class 'ir_isofiles'
c(...)
```

Arguments

`...`	`ir_isofiles` objects to combine

Value

a single combined `ir_isofiles` object

`ir_aggregate_isofiles` *Aggregate data from isofiles*

Description

This function allows dynamic aggregation and validation of data read by `ir_read_isofiles()`. Like `ir_read_isofiles()`, it is designed to be fail save by safely catching errors and reporting back on them (see `ir_get_problems()`). This function should work out of the box for most files without additional modification of the `aggregator`.

Usage

```
ir_aggregate_isofiles(
  isofiles,
  intensity_units = c("mV", "V", "fA", "pA", "nA", "µA", "mA", "A", "cps"),
  aggregator = "standard",
  show_progress = is_interactive(),
  show_problems = TRUE
)
```

Arguments

`isofiles`	the isotope data files/archives read in by `ir_read_isofiles()`
`intensity_units`	target intensity unit to convert traces/cycles/scans to before aggregation, one of `"mV"`, `"V"`, `"fA"`, `"pA"`, `"nA"`, `"\u00b5A"`, `"mA"`, `"A"`, `"cps"`. Only applied when the aggregator includes any of those datasets. Default is `"mV"`.
`aggregator`	typically the name of a registered aggregator (see all with `ir_get_option("aggregators")`), default is the "standard" aggregator included in the package (`ir_get_aggregator("standard")`). The other built-in options, from smallest to largest, are "metadata" (`ir_get_aggregator("metadata")`, metadata only), "minimal" (`ir_get_aggregator("minimal")`, metadata plus the core data series), and "extended" (`ir_get_aggregator("extended")`, also resistors and the vendor data table). The `aggregator` parameter can can also directly be an aggregator tibble (created/modified with `ir_start_aggregator()` and/or `ir_add_to_aggregator()`) that defines which data should be aggregated and how.
`show_progress`	whether to show a progress bar, by default always enabled when running interactively e.g. inside Positron or RStudio (and disabled in a notebook), turn off with `show_progress = FALSE`
`show_problems`	whether to show problems encountered along the way (rather than just keeping track of them with `ir_get_problems()`). Set to `show_problems = FALSE` to turn off the live printout. Either way, all encountered problems can be retrieved with running `ir_get_problems()` for the returned list

Value

a list of merged dataframes collected from the `isofiles` based on the `aggregator` definitions

`ir_calculate_ratios` *Calculate isotope ratios*

Description

Calculate intensity ratios of each mass relative to a base mass for every measurement in an `ir_aggregate_isofiles()` result. Ratios are added directly to the `traces` (continuous flow), `cycles` (dual inlet), and/or `scans` data present in the aggregated data as two extra columns: `ratio_name` (e.g. `"29/28"`) and `ratio`. The ratio at each `time.s`/`cycle`/`x` position (within every `uidx` and `analysis`) is

Usage

```
ir_calculate_ratios(
  aggregated_data,
  ...,
  num_add.V = 100,
  denom_add.V = num_add.V,
  num_add.nA = 0,
  denom_add.nA = num_add.nA,
  num_add.cps = 0,
  denom_add.cps = num_add.cps,
  normalize_ratios = NULL
)
```

Arguments

`aggregated_data`
: datasets aggregated from `ir_aggregate_isofiles()` (must include at least one of `traces`, `cycles`, or `scans`)

`...`
: named base masses for individual species, e.g. `SO2 = 64, N2 = 28`. Species not listed here use their numerically lowest measured mass as the base mass.

`num_add.V`, `denom_add.V`
: additive offset (in volts) for the numerator and denominator intensities of voltage-unit data (`V`/`mV`). Applies to `traces` and `scans` only, not to `cycles` (dual inlet). Default `num_add.V = 100`, `denom_add.V = num_add.V`.

`num_add.nA`, `denom_add.nA`
: additive offset (in nanoamperes) for the numerator and denominator intensities of current-unit data (`A`/`mA`/`µA`/`nA`/`pA`/`fA`). Applies to `traces` and `scans` only, not to `cycles` (dual inlet). Default `num_add.nA = 0`, `denom_add.nA = num_add.nA`.

`num_add.cps`, `denom_add.cps`
: additive offset (in cps) for the numerator and denominator intensities of count-unit data (`cps`). Applies to `traces` and `scans` only, not to `cycles` (dual inlet). Default `num_add.cps = 0`, `denom_add.cps = num_add.cps`.

`normalize_ratios`
: `NULL` (default) for no normalization, or a function (e.g. `mean`, `median`, `min`, `max`) applied per `uidx`/`analysis`/`ratio_name` group; each ratio is divided by the value the function returns for its group's non-`NA` ratios.

Details

$$ratio = (I_{mass} + num_add)/(I_{base} + denom_add)$$

i.e. the intensity of the mass divided by the intensity of the base mass of the same species, after adding an additive offset to numerator and denominator (see below). Base mass rows are kept and have NA in both columns. Calling this function again recomputes (overwrites) the ratio_name/ratio columns. The resulting ratios are not constrained in any way (they can be any value).

The base mass for a species is, by default, the numerically lowest mass measured for that species. Override it for individual species via ... (e.g. SO2 = 64, N2 = 28).

Value

the aggregated_data with ratio_name and ratio columns added to each of the traces, cycles, and/or scans datasets that is present. Both columns are NA for base mass rows (and for any species whose requested base mass could not be found).

Additive offsets

The additive offsets apply to continuous-flow (traces) and scans data **only**. Dual inlet (cycles) data is **not** offset and always uses the plain ratio I_mass / I_base regardless of the num_add.*/denom_add.* settings.

For traces and scans, which pair of additive offsets is used depends on the intensity unit family of the data: voltage (V, mV) uses num_add.V/denom_add.V, current (A, mA, µA, nA, pA, fA) uses num_add.nA/denom_add.nA, and counts (cps) uses num_add.cps/denom_add.cps. The offsets are specified in their family's reference unit (volts, nanoamperes, cps) and are automatically scaled to the data's actual intensity unit before being added. For example, with intensity.mV data the default num_add.V = 100 (volts) is multiplied by 1000 and 100000 mV are added; with intensity.pA data the default num_add.nA = 0 would be multiplied by 1000 (1 nA = 1000 pA).

Normalization

normalize_ratios is NULL by default (no normalization). Pass a function to divide every ratio by the value that function returns for its uidx/analysis/ratio_name group (the function receives the group's non-NA ratios). For example normalize_ratios = mean centers each ratio around 1, while median, min, or max normalize to the group median, minimum, or maximum, respectively.

---

ir_check_isoextract *Check for the isoextract executables*

Description

By default, these will install the executable if it is missing or outdated. They run automatically when needed and do not usually need to be called directly by the user. In particular, ir_check_isoextract() calls ir_check_isosolfs() automatically (unless check_isosolfs = FALSE), so ir_check_isosolfs() rarely needs to be called on its own.

Usage

```
ir_check_isoextract(
  install_if_missing = !on_cran(),
  reinstall_if_outdated = !on_cran(),
  reinstall_always = FALSE,
  min_version = "0.3.3",
  show_version = TRUE,
  ask_permission = TRUE,
  source =
   paste0("https://github.com/isoverse/IsofileExtractor/releases/download/isoextract-v",
    min_version),
  check_isosolfs = TRUE,
  ...
)

ir_check_isosolfs(
  install_if_missing = !on_cran(),
  reinstall_if_outdated = !on_cran(),
  reinstall_always = FALSE,
  min_version = "1.0.0",
  show_version = TRUE,
  ask_permission = TRUE,
  source =
   paste0("https://github.com/isoverse/IsofileExtractor/releases/download/isosolfs-v",
    min_version),
  ...
)
```

Arguments

`install_if_missing`	install the executable if it's missing
`reinstall_if_outdated`	install the executable if it's outdated (i.e. not at least `min_version`)
`reinstall_always`	whether to (re-)install no matter what
`min_version`	the minimum version number required
`show_version`	whether to print the installed version after a successful check (default: `TRUE`)
`ask_permission`	whether to ask for the user's permission before downloading a missing or outdated executable (default: `TRUE`). The prompt only appears in interactive sessions and only when a download is actually needed; if it is declined - or the session is not interactive - no download is attempted and the function aborts with instructions. Set to `FALSE` to allow the download without prompting (e.g. in scripts). When `ir_check_isoextract()` downloads isoextract with the user's consent it passes `ask_permission = FALSE` on to `ir_check_isosolfs()` so the user is not asked a second time.

source	the URL (or local path) where to find the executable, by default this is the latest release of the executables on github
check_isosolfs	whether to also ensure the isosolfs helper executable is installed (default: TRUE), by calling ir_check_isosolfs(). isosolfs is required to read Qtegra notebooks (.imexp files) and is released alongside isoextract; the same install_if_missing / reinstall_if_outdated / reinstall_always / show_version settings are applied to it.
...	passed on to download.file if (re-) installing the executable(s)

Value

called for its side effect of ensuring a working executable (at least min_version) is installed — and, for ir_check_isoextract() when check_isosolfs = TRUE, isosolfs as well; returns NULL invisibly and aborts if a required executable cannot be made available

Functions

- ir_check_isosolfs(): ensure the isosolfs helper executable (used to read Qtegra .imexp notebooks) is installed. Released alongside isoextract and called automatically by ir_check_isoextract(), so it rarely needs to be called directly.

---

ir_convert_intensity *Convert intensity between units*

Description

Note: this function is rarely called directly, it's run as part of ir_aggregate_isofiles to standardize the trace/cycle/scan datasets before aggregation.

Usage

```
ir_convert_intensity(
  dataset,
  resistors = NULL,
  units = c("mV", "V", "fA", "pA", "nA", "µA", "mA", "A", "cps")
)
```

Arguments

dataset	a data frame with columns species, channel, and an intensity.<unit> column (e.g. intensity.mV, intensity.nA).
resistors	a data frame with columns species, channel, and resistance.Ohm. Required when converting between voltage and current/CPS; ignored otherwise. If a config column is present in both dataset and resistors it is included in the join key.
units	target unit, one of "mV", "V", "fA", "pA", "nA", "\u00b5A", "mA", "A", "cps".

Details

Converts the intensity column of a dataset between voltage, current, and count-per-second units using A = CPS * e and V = A * R. Automatically detects the source unit from any intensity.<unit> column present in dataset. Joins resistors by species and channel (plus config when present in both) only when the conversion path crosses the A/V boundary. Returns dataset with the source intensity column replaced by intensity.<units>.

Value

the dataset with its intensity.* column converted to the target units and renamed accordingly (e.g. intensity.mV)

---

ir_default_theme *Default isoreader2 plotting theme*

Description

This theme is always applied by the plotting functions (ir_plot_traces(), ir_plot_dual_inlet(), ir_plot_scans()). To customize a plot, add a ggplot2::theme() on top of the returned plot, e.g. ir_plot_traces(...) + ggplot2::theme(text = element_text(size = 20)).

Usage

```
ir_default_theme(text_size = 16)
```

Arguments

text_size	base font size in points (default: 16)

Value

a ggplot2 theme object

---

ir_examples_folder *Bundled example files*

Description

ir_examples_folder() returns the path to the folder with the example isodat files bundled with the package (a convenience wrapper around system.file("extdata", package = "isoreader2")). ir_copy_examples() copies those example files into a local folder so they can be read, re-extracted, or modified without touching the read-only package installation.

Usage

```
ir_examples_folder()

ir_copy_examples(folder = "examples")
```

Arguments

`folder`	target directory to copy the example files into (default `"examples"`); created if it does not exist

Value

`ir_examples_folder()` returns the path to the example files folder as a single string.

`ir_copy_examples()` invisibly returns the path to the created examples folder

Functions

- `ir_copy_examples()`: copy the bundled example files into a local `folder`, creating it if necessary and only copying files that do not already exist there (existing files are left untouched). Only the original data files are copied, not their bundled `.json` sidecars, so reading from the copied folder re-extracts them from scratch (requires the isoextract executable).

Examples

```
ir_examples_folder() |> ir_find_scans()
# copy the bundled examples into a temporary folder and find them
ir_copy_examples(folder = file.path(tempdir(), "examples")) |>
  ir_find_continuous_flow()
```

`ir_export_to_excel` *Export data to Excel*

Description

Exports one or more data frames / tibbles (typically retrieved with the `ir_get_*()` functions, e.g. `ir_get_metadata()`, `ir_get_traces()`) to an Excel file, one sheet per data frame. Pass the data frames as `...`: **named** arguments use the name as the sheet name, **unnamed** arguments are placed in a sheet named after their position (e.g. the 3rd unnamed data frame goes into `"Sheet3"`).

Usage

```
ir_export_to_excel(
  ...,
  file,
  dbl_digits = 2,
  int_format = "0",
  dbl_format = sprintf(sprintf("%%.%sf", dbl_digits), 0),
  show_progress = is_interactive()
)
```

Arguments

`...`	one or more data frames / tibbles to export, one per sheet. Named arguments set the sheet name; unnamed arguments use `"Sheet{position}"`.
`file`	path to the `.xlsx` file (`.xlsx` extension added if absent)
`dbl_digits`	number of decimal places shown for double columns (all digits are stored; this only affects display formatting in Excel)
`int_format`	Excel number format string for integer columns
`dbl_format`	Excel number format string for double columns (derived automatically from `dbl_digits` if not set)
`show_progress`	whether to show a progress indicator

Details

This function only accepts data frames. To store a complete `ir_aggregate_isofiles()` result use `ir_save_aggregated_data()` instead.

Requires the suggested **openxlsx** package.

Value

the exported data invisibly (the single data frame if one was provided, otherwise the list of data frames), for use in pipes

Examples

```
if (requireNamespace("openxlsx", quietly = TRUE)) {
  agg <- ir_examples_folder() |>
    ir_find_continuous_flow() |>
    ir_read_isofiles() |>
    ir_aggregate_isofiles()
  ir_export_to_excel(
    metadata = ir_get_metadata(agg),
    traces = ir_get_traces(agg),
    file = file.path(tempdir(), "my_export.xlsx")
  )
}
```

`ir_extract_isofiles` *run the isoextract executable on a vector of file paths this is usually not called directly*

Description

run the isoextract executable on a vector of file paths this is usually not called directly

Usage

```
ir_extract_isofiles(
  file_paths,
  pretty_json = FALSE,
  dry_run = FALSE,
  show_progress = is_interactive(),
  show_problems = TRUE
)

ir_get_isoextract_version()
```

Arguments

file_paths	paths to the isodat file(s), single value or vector of paths. Use ir_find_isofiles() to get files in a folder.
pretty_json	whether to write the JSON output in human-readable pretty-printed format (default: FALSE). Useful for debugging; has no effect on the data read back by ir_read_isofiles(). Note that pretty-printed files are larger than compact ones.
dry_run	whether to run isoextract in "dry run" mode (default: FALSE). In dry run mode the files are parsed to test whether they can be read (a file-compatibility check) but no .json sidecar output is written. Combine with show_problems = TRUE to see which files (if any) cannot be extracted. Note that with dry_run = TRUE, the progress bar does not work as it depends on the JSON output files.
show_progress	whether to show a progress bar, by default always enabled when running interactively e.g. inside Positron or RStudio (and disabled in a notebook), turn off with show_progress = FALSE
show_problems	whether to show problems encountered along the way (rather than just keeping track of them with ir_get_problems()). Set to show_problems = FALSE to turn off the live printout. Either way, all encountered problems can be retrieved with running ir_get_problems() for the returned list

Value

called for its side effect of running isoextract to write a .json sidecar file next to each input file (unless dry_run = TRUE); returns NULL invisibly

Functions

- ir_get_isoextract_version(): return the version of the installed isoextract executable as a numeric_version, or NULL if it is not installed (or does not report a recognizable version)

`ir_filter_for` *Filter isofiles by measurement type*

Description

Convenience wrappers around `ir_filter_metadata()` that keep only the files of a single measurement type (using the metadata `type` column): continuous flow (`"cf"`), dual inlet (`"di"`), or scan (`"scan"`). Like `ir_filter_metadata()` they work on both `ir_isofiles` (from `ir_read_isofiles()`) and `ir_aggregated_data` (from `ir_aggregate_isofiles()`) objects, cascade to the other datasets, and drop any file whose metadata ends up empty.

Usage

```
ir_filter_for_continuous_flow(isofiles)

ir_filter_for_dual_inlet(isofiles)

ir_filter_for_scans(isofiles)
```

Arguments

`isofiles`	a collection of isofiles from `ir_read_isofiles()` (`ir_isofiles`) or datasets aggregated from `ir_aggregate_isofiles()` (`ir_aggregated_data`)

Details

Files whose metadata has no `type` column (e.g. a file that errored during reading) never match and are dropped.

Value

the `isofiles` object filtered to the requested measurement type

Functions

- `ir_filter_for_continuous_flow()`: keep only continuous flow files (`type == "cf"`)
- `ir_filter_for_dual_inlet()`: keep only dual inlet files (`type == "di"`)
- `ir_filter_for_scans()`: keep only scan files (`type == "scan"`)

`ir_filter_masses` *Filter isofiles by mass*

Description

Keeps only the requested masses in the `traces`, `cycles`, and `scans` data of either an `ir_aggregate_isofiles()` result (`ir_aggregated_data`) or a collection of isofiles read with `ir_read_isofiles()` (`ir_isofiles`). Like `ir_filter_metadata()` it works on both object types and returns the same type it was given.

Usage

```
ir_filter_masses(isofiles, mass)
```

Arguments

`isofiles`	datasets aggregated from `ir_aggregate_isofiles()` (`ir_aggregated_data`) or a collection of isofiles from `ir_read_isofiles()` (`ir_isofiles`)
`mass`	which masses to keep, as a tidyselect expression evaluated as if the masses present in the data were column names • the same syntax as the `mass` argument of `ir_plot_traces()`. E.g. `c("44", "45")` or `44:48` for specific masses, `-"45"`/`!"45"` to exclude one, `everything()` for all of them, and helpers such as `starts_with("4")`, `matches()`, `all_of()`, or `any_of()`. Unlike plain tidyselect, numbers select by name rather than by position (`44:48` means the masses 44 to 48, not the 44th to 48th mass). Selecting a mass that is not in the data is an error that lists the available masses; use `any_of()` to ignore missing ones.

Details

Any metadata record (`uidx + analysis`) that has no data left in *any* of `traces`/`cycles`/`scans` afterwards is removed from the metadata, and the removal cascades to the other datasets exactly as in `ir_filter_metadata()` (`resistors` and `problems` are restricted to the remaining `uidx`). For an `ir_isofiles` object, a file whose metadata ends up empty is dropped from the collection entirely. Note that this also drops records that never had any `traces`/`cycles`/`scans` data to begin with (e.g. a file that failed to read) - check `ir_get_problems()` before filtering if you need to keep track of those.

The `mass` selection is always resolved **once against the whole object** (all files of an `ir_isofiles` collection, and all of `traces`/`cycles`/`scans` of an aggregated dataset combined), so selecting a mass that only some of the files or datasets contain works and simply leaves the others without data.

Ratios calculated with `ir_calculate_ratios()` live in the `ratio_name` / `ratio` columns of the rows of their **numerator** mass, so they are kept or removed with that mass: filtering to `mass = 44` also removes the `"45/44"` ratio (which sits on the mass 45 rows), while `mass = c(44, 45)` keeps it. `resistors` are never filtered by mass since they describe the instrument configuration rather than measured data.

As for the metadata operations, columns that are entirely `NA` across all remaining rows are dropped from every (non-empty) dataset and the *not-aggregated* column information is cleared.

Value

the `isofiles` object (of the same type as the input) with only the selected masses

Examples

```
## Not run:
# keep only the CO2 masses
dataset |> ir_filter_masses(44:46)

# keep everything except mass 45
dataset |> ir_filter_masses(-"45")

## End(Not run)
```

`ir_find_isofiles` *Find isodat files*

Description

Finds isodat files with the specified extensions in one or more folders.

Usage

```
ir_find_isofiles(
  folder,
  types = c("dxf", "cf", "iarc", "larc", "bch", "imexp", "caf", "did", "scn"),
  pattern = NULL,
  recursive = TRUE
)

ir_find_continuous_flow(folder, pattern = NULL, recursive = TRUE)

ir_find_dual_inlet(folder, pattern = NULL, recursive = TRUE)

ir_find_scans(folder, pattern = NULL, recursive = TRUE)
```

Arguments

`folder`	path to a folder with isodat files, or a character vector of folder paths
`types`	file extensions to include (without leading dot), default is all supported types: `c("dxf", "cf", "iarc", "larc", "bch", "imexp", "caf", "did", "scn")`
`pattern`	provide a name pattern to find only specific files
`recursive`	whether to find files recursively

Value

a sorted character vector of unique paths that correspond to the original data files (without `.json` suffixes if those are the versions of the files that are present)

Functions

- `ir_find_continuous_flow()`: finds continuous flow files (`.dxf`, `.cf`)
- `ir_find_dual_inlet()`: finds dual inlet files (`.did`, `.caf`)
- `ir_find_scans()`: finds scan files (`.scn`)

Examples

```
ir_find_continuous_flow(system.file("extdata", package = "isoreader2"))
ir_find_dual_inlet(system.file("extdata", package = "isoreader2"))
ir_find_scans(system.file("extdata", package = "isoreader2"))
```

---

ir_generate_tibble *Generate the tibble used by the plotting functions*

Description

These helpers build the exact flat tibble that `ir_plot_traces()` (`ir_generate_traces_tibble()`), `ir_plot_dual_inlet()` (`ir_generate_cycles_tibble()`), and `ir_plot_scans()` (`ir_generate_scans_tibble()`) plot, so it can be inspected or used independently of producing a plot. The `dataset` is prepared exactly as for the plotting functions (an `ir_aggregated_data` object has its `traces` / `cycles` / `scans` dataset inner-joined with `$metadata`; a plain data frame is used as is), filtered by `species`, and then split into intensity rows and (optionally) ratio rows, each augmented with four columns:

- `trace` - the identifier `"<species>: <mass>"` for intensity rows (e.g. `"CO2: 44"`) or `"<species>: <ratio_name>"` for ratio rows (e.g. `"CO2: 45/44"`), always (re)generated and returned as a factor sorted by species and numerical (numerator) mass. This is what the plotting functions group their lines by. Rows whose `species` is `NA` get the bare mass/ratio name instead (`"44"`, `"45/44"`) rather than an `"NA: "` prefix.
- `color` - the colour identifier, a factor listing every trace that shares a species and (numerator) mass: both `"CO2: 45"` and `"CO2: 45/44"` get `"CO2: 45, 45/44"` (or `"45, 45/44"` for an `NA` species). Mapping the colour aesthetic to this column (the plotting functions' default) is what draws an intensity trace and its ratios in the same colour while keeping them separate lines. Its levels follow the `trace` order, so the legend runs by ascending species/mass.
- `data_type` - `"intensity [UNITS]"` (e.g. `"intensity [mV]"`) for the intensity rows, or `"ratios"` for ratio rows.
- `value` - the value to plot: the intensity for intensity rows, or the (optionally fold-clamped) ratio for ratio rows.

Usage

```
ir_generate_traces_tibble(
  dataset,
  species = NULL,
  mass = everything(),
  ratio = everything()
)
```

```
ir_generate_cycles_tibble(
  dataset,
  species = NULL,
  mass = everything(),
  ratio = everything()
)

ir_generate_scans_tibble(
  dataset,
  species = NULL,
  mass = everything(),
  ratio = everything()
)
```

Arguments

dataset	an ir_aggregated_data object from ir_aggregate_isofiles() or a plain data frame with the required columns (see the matching plotting function)
species	optional vector to filter to specific species (e.g. "CO2" or c("N2", "CO2")); default NULL keeps all species.
mass	which masses to include as intensity traces, as a tidyselect expression evaluated as if the masses present in the data were column names. The default everything() includes every mass and NULL (or c()) includes none; beyond that any tidyselect syntax works, e.g. c("44", "45") or 44:48 for specific masses, -"45"/!"45" to exclude one, and helpers such as starts_with("4"), matches(), all_of(), or any_of(). Unlike plain tidyselect, numbers select by name rather than by position (44:48 means the masses 44 to 48, not the 44th to 48th mass). Selecting a mass that is not in the data is an error that lists the available masses; use any_of() to ignore missing ones.
ratio	which ratios to include (computed with ir_calculate_ratios()), as a tidyselect expression evaluated as if the ratio names present in the data were column names - the same syntax as mass, e.g. everything() (the default, all available ratios), NULL for none, c("45/44", "46/44"), -"45/44", or starts_with("45"). Selecting specific ratios when ratios have not been calculated is an error pointing to ir_calculate_ratios(); with the default everything() (or NULL) and no ratios present, none are simply added.

Value

a tibble with the prepared data plus the trace, color, data_type, and value columns described above.

`ir_get_data` *Get data frame from aggregated data*

Description

Retrieve a specific subset of the aggregated data into a single data frame by specifying which columns to take from each dataset (metadata, traces, cycles, scans, resistors, vendor_data_table) using `dplyr::select()` syntax. If data from more than one dataset is selected (e.g. some columns from `traces` AND some from `resistors`), the datasets are combined with an `dplyr::inner_join()` using the columns listed in `by` (only the ones actually in the datasets). Joins that would lead to duplicated data entries (i.e. many-to-many joins) are not allowed and will throw an error to avoid unexpected replications of individual datapoints. If you really want to do such a join, you'll have to do it manually.

Usage

```
ir_get_data(
  aggregated_data,
  metadata = c("file_name"),
  traces = NULL,
  cycles = NULL,
  scans = NULL,
  resistors = NULL,
  vendor_data_table = NULL,
  by = c("uidx", "analysis", "config", "species", "channel", "mass")
)

ir_get_metadata(aggregated_data, metadata = everything())

ir_get_resistors(
  aggregated_data,
  metadata = c("file_name"),
  by = c("uidx", "analysis")
)

ir_get_traces(
  aggregated_data,
  metadata = c("file_name"),
  by = c("uidx", "analysis")
)

ir_get_cycles(
  aggregated_data,
  metadata = c("file_name"),
  by = c("uidx", "analysis")
)
```

```
ir_get_scans(
  aggregated_data,
  metadata = c("file_name"),
  by = c("uidx", "config")
)

ir_get_vendor_data_table(
  aggregated_data,
  metadata = c("file_name"),
  by = c("uidx", "analysis")
)
```

Arguments

aggregated_data	datasets aggregated from ir_aggregate_isofiles()
metadata	columns to get from the aggregated metadata, all dplyr::select() syntax is supported
traces	columns to get from the aggregated traces, all dplyr::select() syntax is supported
cycles	columns to get from the aggregated cycles, all dplyr::select() syntax is supported
scans	columns to get from the aggregated scans, all dplyr::select() syntax is supported
resistors	columns to get from the aggregated resistors, all dplyr::select() syntax is supported
vendor_data_table	columns to get from the aggregated vendor_data_table, all dplyr::select() syntax is supported
by	character vector of column names used as join keys when combining data from more than one dataset (default covers the standard linking columns; only keys actually present in both datasets are used)

Value

a tibble

Functions

- ir_get_metadata(): shortcut for retrieving all metadata columns (i.e. metadata = dplyr::everything())
- ir_get_resistors(): shortcut for retrieving all resistors columns (i.e. resistors = dplyr::everything()), keyed by the selected metadata
- ir_get_traces(): shortcut for retrieving all traces columns (i.e. traces = dplyr::everything()), keyed by the selected metadata
- ir_get_cycles(): shortcut for retrieving all cycles columns (i.e. cycles = dplyr::everything()), keyed by the selected metadata

- `ir_get_scans()`: shortcut for retrieving all `scans` columns (i.e. `scans = dplyr::everything()`), keyed by the selected `metadata`
- `ir_get_vendor_data_table()`: shortcut for retrieving all `vendor_data_table` columns (i.e. `vendor_data_table = dplyr::everything()`), keyed by the selected `metadata`. Only available when aggregated with the `"extended"` aggregator.

`ir_get_problems` *Retrieve parsing problems*

Description

This function retrieves parsing problems encountered during the reading and processing of files.

This function prints out parsing problems encountered during the reading and processing of files.

Usage

```
ir_get_problems(obj, strip_ansi = TRUE)

ir_show_problems(obj)
```

Arguments

`obj`	data object that holds problems information
`strip_ansi`	whether to remove ansi characters from the message, yes by default

Value

tibble data frame with a list of problems encountered during processing

---

`ir_get_supported_file_types` *Get supported file types*

Description

Get supported file types

Usage

```
ir_get_supported_file_types()
```

Value

a tibble of the file types supported by this package

Examples

```
ir_get_supported_file_types()
```

`ir_isofiles_storage` *Save and load isofiles*

Description

`ir_save_isofiles()` serializes a collection of isofiles read with `ir_read_isofiles()` to an RDS file using `readr::write_rds()`, storing the whole `ir_isofiles` object as-is (including all nested datasets and condition objects) without any changes. `ir_load_isofiles()` reads the file back with `readr::read_rds()` and returns the `ir_isofiles` object exactly as it was saved.

Usage

```
ir_save_isofiles(isofiles, file)

ir_load_isofiles(file)
```

Arguments

`isofiles`	a collection of isofiles from `ir_read_isofiles()`
`file`	path to the RDS file (`.rds` extension added if absent)

Details

This operates at the unaggregated `ir_isofiles` level. To store an aggregated result instead, use `ir_save_aggregated_data()` / `ir_load_aggregated_data()`.

Value

`ir_save_isofiles()` returns `isofiles` invisibly; `ir_load_isofiles()` returns an `ir_isofiles` object.

Functions

- `ir_save_isofiles()`: save isofiles to an RDS file
- `ir_load_isofiles()`: load isofiles from an RDS file

ir_metadata *Filter, mutate, or join the metadata of isofiles*

Description

These functions modify the metadata of either an `ir_aggregate_isofiles()` result (`ir_aggregated_data`) or a collection of isofiles read with `ir_read_isofiles()` (`ir_isofiles`).

Usage

```
ir_filter_metadata(isofiles, ...)

ir_mutate_metadata(isofiles, ...)

ir_join_metadata(isofiles, y, by)
```

Arguments

`isofiles`	datasets aggregated from `ir_aggregate_isofiles()` (`ir_aggregated_data`) or a collection of isofiles from `ir_read_isofiles()` (`ir_isofiles`)
`...`	passed to `dplyr::filter()`, `dplyr::mutate()`, or `dplyr::left_join()` respectively
`y`	data frame to join to the metadata
`by`	character vector of columns to join by (passed to `dplyr::left_join()`)

Details

For `ir_aggregated_data`, the operation is applied once to the combined `$metadata` data frame. For `ir_filter_metadata()`, the filter then cascades to all other datasets: `traces`, `cycles`, `scans`, and `vendor_data_table` (if aggregated) are filtered by the remaining `uidx` + `analysis` combinations; `resistors` and `problems` are filtered by the remaining `uidx` values.

For `ir_isofiles`, the same operation is instead applied **individually to each row** (i.e. to each file's own nested datasets), since an `ir_isofiles` object has no combined metadata to operate on. Within each row, the filter cascade uses whichever linking columns are present (typically `analysis`). For `ir_filter_metadata()`, any file whose metadata ends up with 0 rows after the filter is removed from the `ir_isofiles` collection entirely.

Operating on an unaggregated `ir_isofiles` object is supported for convenience, but is **significantly slower** than operating on an `ir_aggregated_data` result, because the operation has to be carried out separately on every file rather than once on the combined metadata. For anything beyond small collections, prefer aggregating first with `ir_aggregate_isofiles()` and then applying these functions to the result.

After filtering, columns that are entirely `NA` across all remaining rows are dropped from every (non-empty) dataset. All three functions also clear the *not-aggregated* column information (columns present in the source files but not included in the aggregator) from every dataset, since that information is no longer meaningful after the metadata has been modified.

Value

the `isofiles` object (of the same type as the input) with updated metadata

Functions

- `ir_filter_metadata()`: filter rows of the metadata (and cascade to the other datasets)
- `ir_mutate_metadata()`: add or modify columns in the metadata
- `ir_join_metadata()`: left-join additional columns into the metadata

`ir_options` *Package options*

Description

These options are best set via `ir_options()` and queried via `ir_get_option()`. However, the base functions `options()` and `getOption()` work as well but require an `isoreader2.` prefix (the package name and a dot) for the option name. Setting an option to a value of `NULL` means that the default is used. `ir_get_options()` is available as an additional convenience function to retrieve a subset of options with a regular expression pattern.

Usage

```
ir_options(...)

ir_get_options(pattern = NULL)

ir_get_option(x)
```

Arguments

`...`	set package options, syntax identical to `options()`
`pattern`	to retrieve multiple options (as a list) with a shared pattern
`x`	name of the specific option to retrieve

Value

`ir_options()` and `ir_get_options()` return a named list of option values; `ir_get_option()` returns the value of the single requested option.

Functions

- `ir_options()`: set/get option values
- `ir_get_options()`: get a subset of option values that fit a pattern
- `ir_get_option()`: retrieve the current value of one option (option must be defined for the package)

Options for the isoreader2 package

- `aggregators`: data aggregators for pulling data out of raw files. The list of available aggregators is accessible via `ir_get_option("aggregators")`. Individual aggregators are available via the shortcut helper function `ir_get_aggregator("standard")`. Register new/overwrite existing aggregators via `ir_register_aggregator()`.
- `debug`: turn on debug mode
- `auto_use_ansi`: whether to automatically enable correct rendering of stylized (ansi) output in HTML reports from notebooks that call `library(isoir)`. Can be turned off by calling `isoir::ir_options(auto_use_ansi = FALSE)` **before** call `library(isoir)`.

Examples

```
# All default options
ir_get_options()
```

`ir_plot_dual_inlet` *Plot dual inlet cycle data*

Description

Plots cycle data from an `ir_aggregate_isofiles()` result or a plain data frame. The data is prepared with `ir_generate_cycles_tibble()` (which, for an `ir_aggregated_data` object, inner-joins the `$cycles` dataset with `$metadata`). The plot data must contain `species`, `cycle`, `type`, `mass`, and an `intensity.*` column — an error is thrown if any are missing. A `trace` identifier (`"<species>: <mass>"`) and the matching `color` identifier are always regenerated and the plotted `value` together with a `data_type` label (`"intensity [UNITS]"`, or `"ratios"` for ratio rows) are added.

Usage

```
ir_plot_dual_inlet(
  dataset,
  species = NULL,
  mass = everything(),
  ratio = everything(),
  facet = NULL,
  data_type_as_facet = auto(),
  scales = "free",
  nrow = NULL,
  ncol = 1,
  color = color,
  shape = type,
  linetype = NULL,
  color_values = palette.colors(),
  drop_unused_levels = FALSE,
```

```
  scientific = FALSE,
  cycle_window = NULL,
  n_y_breaks = 5,
  ...
)
```

Arguments

dataset	an ir_aggregated_data object from ir_aggregate_isofiles() or a plain data frame with species, cycle, type, mass, and an intensity.* column
species	optional vector to filter the displayed data to specific species (e.g. "CO2" or c("N2", "CO2")); default NULL shows all species.
mass	which masses to show as intensity traces, as a tidyselect expression evaluated as if the masses present in the data were column names. The default everything() shows every mass and NULL (or c()) shows none; beyond that any tidyselect syntax works, e.g. c("44", "45") or 44:48 for specific masses, -"45"/!"45" to exclude one, and helpers such as starts_with("4"), matches(), all_of(), or any_of(). Unlike plain tidyselect, numbers select by name rather than by position (44:48 means the masses 44 to 48, not the 44th to 48th mass). Selecting a mass that is not in the data is an error that lists the available masses; use any_of() to ignore missing ones.
ratio	which ratios to additionally show (computed with ir_calculate_ratios()), as a tidyselect expression evaluated as if the ratio names present in the data were column names - the same syntax as mass, e.g. everything() (the default, all available ratios), NULL for none, c("45/44", "46/44"), -"45/44", or starts_with("45"). Selecting specific ratios when ratios have not been calculated is an error pointing to ir_calculate_ratios() (with the default everything(), or NULL, and no ratios present, none are simply added). Ratio rows are plotted on the same value axis with data_type = "ratios"; data_type is then used as a facet row (see data_type_as_facet) to separate them from the intensities.
facet	column or expression to facet by (default: NULL, no extra faceting). When data_type is used as a facet row (see data_type_as_facet), a single facet variable becomes the facet_grid column (data_type ~ facet) and a NULL facet gives data_type ~ .. Otherwise a plain column or expression (e.g. file_name or paste(species, mass)) is faceted with ggplot2::facet_wrap(), and a two-sided formula (e.g. species ~ mass) is faceted with ggplot2::facet_grid(). Set to NULL to suppress faceting.
data_type_as_facet	whether the data_type column (intensities vs ratios) is used as the ggplot2::facet_grid() row variable: auto() (default) uses it only when more than one data type is present; TRUE always uses it; FALSE never does. When used, the y axis label is dropped (the facet strip provides it) and the facet becomes data_type ~ . (a NULL facet) or data_type ~ facet (a single-variable facet). It is ignored when facet is a two-sided formula (a warning is issued if data_type_as_facet = TRUE is combined with a formula facet, since the two are mutually exclusive).
scales	whether facet scales should be "free" (default), "fixed", "free_x", or "free_y"; passed on to ggplot2::facet_wrap() / ggplot2::facet_grid().

`nrow`, `ncol` — number of rows and columns of facet panels (`nrow` default `NULL` lets ggplot2 choose; `ncol` default `1` stacks the panels in a single column). Only applies when `facet` is a single variable or expression (faceted with `ggplot2::facet_wrap()`); ignored when `facet` is a formula (faceted with `ggplot2::facet_grid()`), with a warning if you set them explicitly.

`color` — column or expression for the colour aesthetic (default: the generated `color` column, which holds all the traces that share a species and (numerator) mass, e.g. `"N2: 29, 29/28"` for both the intensity trace `"N2: 29"` and its ratio trace `"N2: 29/28"`, so they are drawn in the same colour while remaining separate lines). The legend for it is labelled `trace`; use `color = trace` to give every trace its own colour instead.

`shape` — column or expression for the point shape aesthetic (default: `type`, distinguishing `"standard"` from `"sample"` cycles)

`linetype` — column or expression for the linetype aesthetic (default: `NULL`, i.e. no linetype aesthetic)

`color_values` — named or unnamed character vector of colours passed to `ggplot2::scale_color_manual()`, or `NULL` to use the ggplot2 default colour palette (default: `palette.colors()`)

`drop_unused_levels` — whether to drop unused colour factor levels (e.g. traces that are absent after zooming to a window) from the colour scale and legend. Default `FALSE` keeps every level so the colour mapping stays stable across subsets of the same dataset; set to `TRUE` to show only the levels actually present in the plotted data. Note that a `color` level covers every trace of its species/mass, so it is only dropped when none of them is shown.

`scientific` — whether to format y axis labels in scientific notation (default: `FALSE`)

`cycle_window` — optional numeric vector of length 2 giving the cycle axis display window `c(min, max)` (must have `min < max`). The data point just outside each edge of the window is retained so the clipped lines interpolate correctly across the window boundaries and y autoscales correctly at the edges; `ggplot2::coord_cartesian()` clips the display. A window that contains no data points of its own is allowed (the line is drawn between the bracketing points). Default `NULL` shows all cycles.

`n_y_breaks` — desired number of y axis tick marks (default: `5`)

`...` — additional arguments passed on to `ggplot2::facet_wrap()` or `ggplot2::facet_grid()` (e.g. `labeller`)

Value

a `ggplot` object with `ir_default_theme()` applied. To customize the plot, add ggplot2 layers on top (e.g. `+ ggplot2::theme(...)` or `+ ggplot2::labs(...)`); attach ggplot2 with `library(ggplot2)` first.

`ir_plot_scans` *Plot scan data*

Description

Plots scan data from an `ir_aggregate_isofiles()` result or a plain data frame. The data is prepared with `ir_generate_scans_tibble()` (which, for an `ir_aggregated_data` object, inner-joins the `$scans` dataset with `$metadata`). The plot data must contain `species`, `x`, `scan_type`, `x_units`, `mass`, and an `intensity.*` column — an error is thrown if any are missing. A `trace` identifier (`"<species>: <mass>"`) and the matching `color` identifier are always regenerated and the plotted `value` together with a `data_type` label (`"intensity [UNITS]"`, or `"ratios"` for ratio rows) are added. `scan_type` and `x_units` are combined for the x axis label.

Usage

```
ir_plot_scans(
  dataset,
  scan_type = NULL,
  species = NULL,
  mass = everything(),
  ratio = everything(),
  facet = NULL,
  data_type_as_facet = auto(),
  scales = "free",
  nrow = NULL,
  ncol = 1,
  color = color,
  linetype = NULL,
  color_values = palette.colors(),
  drop_unused_levels = FALSE,
  scientific = FALSE,
  x_window = NULL,
  n_x_breaks = 5,
  n_y_breaks = 5,
  ...
)
```

Arguments

`dataset`	an `ir_aggregated_data` object from `ir_aggregate_isofiles()` or a plain data frame with `species`, `x`, `scan_type`, `x_units`, `mass`, and an `intensity.*` column
`scan_type`	which scan type to plot (e.g. `"high voltage"`). Required when the data contains more than one scan type; an error lists the available types. If the data contains only one scan type, the parameter must either be `NULL` or match that type exactly.
`species`	optional vector to filter the displayed data to specific species (e.g. `"CO2"` or `c("N2", "CO2")`); default `NULL` shows all species.

`mass` which masses to show as intensity traces, as a tidyselect expression evaluated as if the masses present in the data were column names. The default `everything()` shows every mass and `NULL` (or `c()`) shows none; beyond that any tidyselect syntax works, e.g. `c("44", "45")` or `44:48` for specific masses, `-"45"/!"45"` to exclude one, and helpers such as `starts_with("4")`, `matches()`, `all_of()`, or `any_of()`. Unlike plain tidyselect, numbers select by name rather than by position (`44:48` means the masses 44 to 48, not the 44th to 48th mass). Selecting a mass that is not in the data is an error that lists the available masses; use `any_of()` to ignore missing ones.

`ratio` which ratios to additionally show (computed with `ir_calculate_ratios()`), as a tidyselect expression evaluated as if the ratio names present in the data were column names - the same syntax as `mass`, e.g. `everything()` (the default, all available ratios), `NULL` for none, `c("45/44", "46/44")`, `-"45/44"`, or `starts_with("45")`. Selecting specific ratios when ratios have not been calculated is an error pointing to `ir_calculate_ratios()` (with the default `everything()`, or `NULL`, and no ratios present, none are simply added). Ratio rows are plotted on the same `value` axis with `data_type = "ratios"`; `data_type` is then used as a facet row (see `data_type_as_facet`) to separate them from the intensities.

`facet` column or expression to facet by (default: `NULL`, no extra faceting). When `data_type` is used as a facet row (see `data_type_as_facet`), a single `facet` variable becomes the facet_grid column (`data_type ~ facet`) and a `NULL` facet gives `data_type ~ .`. Otherwise a plain column or expression (e.g. `file_name` or `paste(species, mass)`) is faceted with `ggplot2::facet_wrap()`, and a two-sided formula (e.g. `species ~ mass`) is faceted with `ggplot2::facet_grid()`. Set to `NULL` to suppress faceting.

`data_type_as_facet` whether the `data_type` column (intensities vs ratios) is used as the `ggplot2::facet_grid()` row variable: `auto()` (default) uses it only when more than one data type is present; `TRUE` always uses it; `FALSE` never does. When used, the y axis label is dropped (the facet strip provides it) and the facet becomes `data_type ~ .` (a `NULL` `facet`) or `data_type ~ facet` (a single-variable `facet`). It is ignored when `facet` is a two-sided formula (a warning is issued if `data_type_as_facet = TRUE` is combined with a formula `facet`, since the two are mutually exclusive).

`scales` whether facet scales should be `"free"` (default), `"fixed"`, `"free_x"`, or `"free_y"`; passed on to `ggplot2::facet_wrap()` / `ggplot2::facet_grid()`.

`nrow, ncol` number of rows and columns of facet panels (`nrow` default `NULL` lets ggplot2 choose; `ncol` default `1` stacks the panels in a single column). Only applies when `facet` is a single variable or expression (faceted with `ggplot2::facet_wrap()`); ignored when `facet` is a formula (faceted with `ggplot2::facet_grid()`), with a warning if you set them explicitly.

`color` column or expression for the colour aesthetic (default: the generated `color` column, which holds all the traces that share a species and (numerator) mass, e.g. `"N2: 29, 29/28"` for both the intensity trace `"N2: 29"` and its ratio trace `"N2: 29/28"`, so they are drawn in the same colour while remaining separate lines). The legend for it is labelled `trace`; use `color = trace` to give every trace its own colour instead.

linetype	column or expression for the linetype aesthetic (default: NULL, i.e. no linetype aesthetic)
color_values	named or unnamed character vector of colours passed to ggplot2::scale_color_manual(), or NULL to use the ggplot2 default colour palette (default: palette.colors())
drop_unused_levels	whether to drop unused colour factor levels (e.g. traces that are absent after zooming to a window) from the colour scale and legend. Default FALSE keeps every level so the colour mapping stays stable across subsets of the same dataset; set to TRUE to show only the levels actually present in the plotted data. Note that a color level covers every trace of its species/mass, so it is only dropped when none of them is shown.
scientific	whether to format y axis labels in scientific notation (default: FALSE)
x_window	optional numeric vector of length 2 giving the x axis display window c(min, max) (must have min < max). The data point just outside each edge of the window is retained so the clipped lines interpolate correctly across the window boundaries and y autoscales correctly at the edges; ggplot2::coord_cartesian() clips the display. A window that contains no data points of its own is allowed (the line is drawn between the bracketing points). Default NULL shows the full x range.
n_x_breaks	desired number of x axis tick marks (default: 5)
n_y_breaks	desired number of y axis tick marks (default: 5)
...	additional arguments passed on to ggplot2::facet_wrap() or ggplot2::facet_grid() (e.g. labeller)

Value

a ggplot object with ir_default_theme() applied. To customize the plot, add ggplot2 layers on top (e.g. + ggplot2::theme(...) or + ggplot2::labs(...)); attach ggplot2 with library(ggplot2) first.

`ir_plot_traces` *Plot trace data*

Description

Plots chromatographic trace data (continuous flow) from an ir_aggregate_isofiles() result or a plain data frame. The data is prepared with ir_generate_traces_tibble() (which, for an ir_aggregated_data object, inner-joins the $traces dataset with $metadata). The plot data must contain species, time.s, mass, and an intensity.* column — an error is thrown if any are missing. A trace identifier ("<species>: <mass>") and the matching color identifier are always regenerated and the plotted value together with a data_type label ("intensity [UNITS]", or "ratios" for ratio rows) are added.

ir_plot_continuous_flow() is a deprecated alias for ir_plot_traces() - it takes the same arguments and returns the same plot. Use ir_plot_traces() instead, it is the more flexible name for what the function actually plots (trace data, which continuous flow files contain).

Usage

```
ir_plot_traces(
  dataset,
  species = NULL,
  mass = everything(),
  ratio = everything(),
  facet = NULL,
  data_type_as_facet = auto(),
  scales = "free",
  nrow = NULL,
  ncol = 1,
  color = color,
  linetype = NULL,
  color_values = palette.colors(),
  drop_unused_levels = FALSE,
  scientific = FALSE,
  time_window.s = if (is.null(time_window.min)) NULL else 60 * time_window.min,
  time_window.min = NULL,
  short_time_labels = FALSE,
  n_time_breaks = 5,
  n_y_breaks = 5,
  ...
)

ir_plot_continuous_flow(...)
```

Arguments

dataset	an ir_aggregated_data object from ir_aggregate_isofiles() or a plain data frame with species, time.s, mass, and an intensity.* column
species	optional vector to filter the displayed data to specific species (e.g. "CO2" or c("N2", "CO2")); default NULL shows all species.
mass	which masses to show as intensity traces, as a tidyselect expression evaluated as if the masses present in the data were column names. The default everything() shows every mass and NULL (or c()) shows none; beyond that any tidyselect syntax works, e.g. c("44", "45") or 44:48 for specific masses, -"45"/!"45" to exclude one, and helpers such as starts_with("4"), matches(), all_of(), or any_of(). Unlike plain tidyselect, numbers select by name rather than by position (44:48 means the masses 44 to 48, not the 44th to 48th mass). Selecting a mass that is not in the data is an error that lists the available masses; use any_of() to ignore missing ones.
ratio	which ratios to additionally show (computed with ir_calculate_ratios()), as a tidyselect expression evaluated as if the ratio names present in the data were column names - the same syntax as mass, e.g. everything() (the default, all available ratios), NULL for none, c("45/44", "46/44"), -"45/44", or starts_with("45"). Selecting specific ratios when ratios have not been calculated is an error pointing to ir_calculate_ratios() (with the default

everything(), or NULL, and no ratios present, none are simply added). Ratio rows are plotted on the same value axis with data_type = "ratios"; data_type is then used as a facet row (see data_type_as_facet) to separate them from the intensities.

facet — column or expression to facet by (default: NULL, no extra faceting). When data_type is used as a facet row (see data_type_as_facet), a single facet variable becomes the facet_grid column (data_type ~ facet) and a NULL facet gives data_type ~ .. Otherwise a plain column or expression (e.g. file_name or paste(species, mass)) is faceted with ggplot2::facet_wrap(), and a two-sided formula (e.g. species ~ mass) is faceted with ggplot2::facet_grid(). Set to NULL to suppress faceting.

data_type_as_facet — whether the data_type column (intensities vs ratios) is used as the ggplot2::facet_grid() row variable: auto() (default) uses it only when more than one data type is present; TRUE always uses it; FALSE never does. When used, the y axis label is dropped (the facet strip provides it) and the facet becomes data_type ~ . (a NULL facet) or data_type ~ facet (a single-variable facet). It is ignored when facet is a two-sided formula (a warning is issued if data_type_as_facet = TRUE is combined with a formula facet, since the two are mutually exclusive).

scales — whether facet scales should be "free" (default), "fixed", "free_x", or "free_y"; passed on to ggplot2::facet_wrap() / ggplot2::facet_grid().

nrow, ncol — number of rows and columns of facet panels (nrow default NULL lets ggplot2 choose; ncol default 1 stacks the panels in a single column). Only applies when facet is a single variable or expression (faceted with ggplot2::facet_wrap()); ignored when facet is a formula (faceted with ggplot2::facet_grid()), with a warning if you set them explicitly.

color — column or expression for the colour aesthetic (default: the generated color column, which holds all the traces that share a species and (numerator) mass, e.g. "N2: 29, 29/28" for both the intensity trace "N2: 29" and its ratio trace "N2: 29/28", so they are drawn in the same colour while remaining separate lines). The legend for it is labelled trace; use color = trace to give every trace its own colour instead.

linetype — column or expression for the linetype aesthetic (default: NULL, i.e. no linetype aesthetic)

color_values — named or unnamed character vector of colours passed to ggplot2::scale_color_manual(), or NULL to use the ggplot2 default colour palette (default: palette.colors())

drop_unused_levels — whether to drop unused colour factor levels (e.g. traces that are absent after zooming to a window) from the colour scale and legend. Default FALSE keeps every level so the colour mapping stays stable across subsets of the same dataset; set to TRUE to show only the levels actually present in the plotted data. Note that a color level covers every trace of its species/mass, so it is only dropped when none of them is shown.

scientific — whether to format y axis labels in scientific notation (default: FALSE)

time_window.s, time_window.min — optional numeric vector of length 2 giving the time axis display window c(min, max), either in seconds (time_window.s) or in minutes (time_window.min,

converted to seconds internally — the function always works in seconds). Provide at most one; if both are given, `time_window.s` is used. Must have `min < max`. The data point just outside each edge of the window is retained so the clipped lines interpolate correctly across the window boundaries and y autoscales correctly at the edges; `ggplot2::coord_cartesian()` clips the display. A window that contains no data points of its own is allowed (the line is drawn between the bracketing points). Default `NULL` (both) shows the full time range.

`short_time_labels`
: whether to use compact time axis labels with no space between value and unit and abbreviated units (`hr`, `m`, `s`) (default: `FALSE`)

`n_time_breaks`
: desired number of time axis tick marks (default: `5`)

`n_y_breaks`
: desired number of y axis tick marks (default: `5`)

`...`
: additional arguments passed on to `ggplot2::facet_wrap()` or `ggplot2::facet_grid()` (e.g. `labeller`); for the deprecated `ir_plot_continuous_flow()` all arguments are simply passed on to `ir_plot_traces()`

Value

a `ggplot` object with `ir_default_theme()` applied. To customize the plot, add ggplot2 layers on top (e.g. `+ ggplot2::theme(...)` or `+ ggplot2::labs(...)`); attach ggplot2 with `library(ggplot2)` first.

---

ir_read_isofiles *Read isotope data files*

Description

Read isotope data files

Usage

```
ir_read_isofiles(
  file_paths,
  show_progress = is_interactive(),
  show_problems = TRUE,
  reextract = FALSE
)
```

Arguments

`file_paths`
: paths to the isodat file(s), single value or vector of paths. Use `ir_find_isofiles()` to get files in a folder.

`show_progress`
: whether to show a progress bar, by default always enabled when running interactively e.g. inside Positron or RStudio (and disabled in a notebook), turn off with `show_progress = FALSE`

show_problems — whether to show problems encountered along the way (rather than just keeping track of them with ir_get_problems()). Set to show_problems = FALSE to turn off the live printout. Either way, all encountered problems can be retrieved with running ir_get_problems() for the returned list

reextract — whether to re-extract files (uses isoextract to read files from scratch); if FALSE (default) only files that have not been extracted yet (or whose previous extraction is out of date) are extracted. Use ir_extract_isofiles() directly for finer control over extraction (e.g. pretty_json or dry_run).

Value

a tibble data frame (an ir_isofiles object) where each row holds the file path and nested tibbles of datasets extracted from the isodat files. Use ir_aggregate_isofiles() to aggregate data safely across files. Multiple such collections can be combined into one with a simple c() (see c.ir_isofiles()).

See Also

c.ir_isofiles() to combine collections of isofiles

---

ir_start_aggregator *Dynamic data aggregator*

Description

These functions allow definition of custom data aggregators for processing data extracted from isofiles. An aggregator is run on each imported file and pulls together the relevant data users are interested in while making sure data formats are correct so that the aggregated data can be merged across several imported files for fast downstream processing.

Usage

```
ir_start_aggregator(name)

ir_add_to_aggregator(
  aggregator,
 dataset = c("metadata", "traces", "cycles", "scans", "resistors", "vendor_data_table"),
  column,
  source = column,
  default = NA,
  cast = "as.character",
  regexp = FALSE,
  func = NULL,
  args = NULL
)

ir_remove_from_aggregator(
```

```
  aggregator,
 dataset = c("metadata", "traces", "cycles", "scans", "resistors", "vendor_data_table"),
  column = NULL
)

ir_register_aggregator(aggregator, name = attr(aggregator, "name"))

ir_get_aggregator(name)
```

Arguments

name	a descriptive name for the aggregator. This name is automatically used as the default name when registering the aggregator via ir_register_aggregator().
aggregator	the aggregator table generated by ir_start_aggregator() or passed from a previous call to ir_add_to_aggregator() for constructing the entire aggregator by piping
dataset	the name of the dataset to aggregate from, by default "metadata" which is by far the most common aggregator to work with
column	the name of the column in which data should be stored. For ir_remove_from_aggregator(), the (optional) name of the column whose aggregator entry should be removed from the dataset - if not provided, all entries for the dataset are removed.
source	single character column name or vector of column names (if alternatives could be the source) where in the dataset to find data for the column. If a vector of multiple column names is provided (e.g. source = c("a1", "a2")), the first column name that's found during processing of a dataset will be used and passed to the function defined in func (if any) and then the one defined in cast. To provide multiple parameters from the data to func, define a list instead of a vector source = list("a", "b", "c") or if multiple alternative columns can be the source for any of the arguments, define as source = list(c("a1", "a2"), "b", c("c1", "c2", "c3"))
default	the default value if no source columns can be found or another error is encountered during aggregation. Note that the default value will also be processed with the function in cast to make sure it has the correct data type.
cast	what to cast the values of the resulting column to, most commonly "as.character", "as.integer", "as.numeric", or "as.factor". This is required to ensure all aggregated values have the correct data type.
regexp	whether source column names should be interpreted as a regular expressions for the purpose of finding the relevant column(s). Note if regexp = TRUE, the search for the source column always becomes case-insensitive so this can also be used for a direct match of a source column whose upper/lower casing can be unreliable. If a column is matched by a regexp and also by a direct aggregator rule, the direct aggregator rule takes precedence.
func	name of a processing function to apply before casting the value with the cast function. This is optional and can be used to conduct more elaborate preprocessing of a data or combining data from multiple source columns in the correct way (e.g. pasting together from multiple columns).

`args` an optional list of arguments to pass to the `func` in addition to the values coming from the source column(s)

Value

an aggregator tibble

Functions

- `ir_start_aggregator()`: starts the aggregator
- `ir_add_to_aggregator()`: add additional column to aggregate data for. Overwrites an existing aggregator entry for the same dataset and column if it already exists.
- `ir_remove_from_aggregator()`: remove an entry (`dataset + column`) or all entries of a `dataset` from the aggregator. Warns if there is nothing to remove.
- `ir_register_aggregator()`: register an aggregator in the isoreader2 options so it can be retrieved with `ir_get_aggregator()`
- `ir_get_aggregator()`: retrieve a registered aggregator (get all aggregators with `ir_get_option("aggregators")`)

---

ir_storage *Save and load aggregated isofile data*

Description

`ir_save_aggregated_data()` serializes an `ir_aggregate_isofiles()` result to a parquet file. Empty datasets (no columns) are dropped. The `condition` column of `problems` is set to `NULL` per row because R condition objects cannot be stored in parquet. `ir_load_aggregated_data()` reads the file back and returns an `ir_aggregated_data` object.

Usage

```
ir_save_aggregated_data(aggregated_data, file)

ir_load_aggregated_data(file)
```

Arguments

`aggregated_data` datasets aggregated from `ir_aggregate_isofiles()`

`file` path to the parquet file (`.parquet` extension added if absent)

Details

Requires the suggested **arrow** package.

Value

`ir_save_aggregated_data()` returns `aggregated_data` invisibly; `ir_load_aggregated_data()` returns an `ir_aggregated_data` object.

Functions

- `ir_save_aggregated_data()`: save aggregated data to a parquet file
- `ir_load_aggregated_data()`: load aggregated data from a parquet file

Index

auto, 3

c(), *34*
c.ir_aggregated_data, 4
c.ir_isofiles, 4
c.ir_isofiles(), *34*

dplyr::bind_rows(), *4*
dplyr::filter(), *23*
dplyr::inner_join(), *19*
dplyr::left_join(), *23*
dplyr::mutate(), *23*
dplyr::select(), *19, 20*

get_pkg_options *(*ir_options*)*, 24
getOption(), *24*
ggplot2::coord_cartesian(), *27*, *30*, *33*
ggplot2::facet_grid(), *26, 27*, *29, 30*, *32, 33*
ggplot2::facet_wrap(), *26, 27*, *29, 30*, *32, 33*
ggplot2::scale_color_manual(), *27*, *30*, *32*
ggplot2::theme(), *10*

ir_add_to_aggregator *(*ir_start_aggregator*)*, 34
ir_add_to_aggregator(), *5*, *35*
ir_aggregate_isofiles, 5
ir_aggregate_isofiles(), *4*, *6*, *12*, *14, 15*, *18*, *20*, *23*, *25, 26*, *28*, *30, 31*, *34*, *36*
ir_calculate_ratios, 6
ir_calculate_ratios(), *15*, *18*, *26*, *29*, *31*
ir_check_isoextract, 7
ir_check_isosolfs *(*ir_check_isoextract*)*, 7
ir_check_isosolfs(), *9*
ir_convert_intensity, 9
ir_copy_examples *(*ir_examples_folder*)*, 10
ir_default_theme, 10
ir_default_theme(), *27*, *30*, *33*
ir_examples_folder, 10
ir_export_to_excel, 11
ir_extract_isofiles, 12
ir_extract_isofiles(), *34*
ir_filter_for, 14
ir_filter_for_continuous_flow *(*ir_filter_for*)*, 14
ir_filter_for_dual_inlet *(*ir_filter_for*)*, 14
ir_filter_for_scans *(*ir_filter_for*)*, 14
ir_filter_masses, 15
ir_filter_metadata *(*ir_metadata*)*, 23
ir_filter_metadata(), *14, 15*
ir_find_continuous_flow *(*ir_find_isofiles*)*, 16
ir_find_dual_inlet *(*ir_find_isofiles*)*, 16
ir_find_isofiles, 16
ir_find_isofiles(), *13*, *33*
ir_find_scans *(*ir_find_isofiles*)*, 16
ir_generate_cycles_tibble *(*ir_generate_tibble*)*, 17
ir_generate_cycles_tibble(), *25*
ir_generate_scans_tibble *(*ir_generate_tibble*)*, 17
ir_generate_scans_tibble(), *28*
ir_generate_tibble, 17
ir_generate_traces_tibble *(*ir_generate_tibble*)*, 17
ir_generate_traces_tibble(), *30*
ir_get_aggregator *(*ir_start_aggregator*)*, 34
ir_get_aggregator(), *36*
ir_get_cycles *(*ir_get_data*)*, 19
ir_get_data, 19
ir_get_isoextract_version *(*ir_extract_isofiles*)*, 12

ir_get_metadata *(*ir_get_data*)*, 19
ir_get_metadata(), *11*
ir_get_option *(*ir_options*)*, 24
ir_get_option(), *24*
ir_get_options *(*ir_options*)*, 24
ir_get_options(), *24*
ir_get_problems, 21
ir_get_problems(), *5*, *13*, *15*, *34*
ir_get_resistors *(*ir_get_data*)*, 19
ir_get_scans *(*ir_get_data*)*, 19
ir_get_supported_file_types, 21
ir_get_traces *(*ir_get_data*)*, 19
ir_get_traces(), *11*
ir_get_vendor_data_table *(*ir_get_data*)*, 19
ir_isofiles_storage, 22
ir_join_metadata *(*ir_metadata*)*, 23
ir_load_aggregated_data *(*ir_storage*)*, 36
ir_load_aggregated_data(), *22*
ir_load_isofiles *(*ir_isofiles_storage*)*, 22
ir_metadata, 23
ir_mutate_metadata *(*ir_metadata*)*, 23
ir_options, *3*, 24
ir_options(), *24*
ir_plot_continuous_flow *(*ir_plot_traces*)*, 30
ir_plot_dual_inlet, 25
ir_plot_dual_inlet(), *3*, *10*, *17*
ir_plot_scans, 28
ir_plot_scans(), *3*, *10*, *17*
ir_plot_traces, 30
ir_plot_traces(), *3*, *10*, *15*, *17*
ir_read_isofiles, 33
ir_read_isofiles(), *4, 5*, *13–15*, *22, 23*
ir_register_aggregator *(*ir_start_aggregator*)*, 34
ir_register_aggregator(), *35*
ir_remove_from_aggregator *(*ir_start_aggregator*)*, 34
ir_save_aggregated_data *(*ir_storage*)*, 36
ir_save_aggregated_data(), *12*, *22*
ir_save_isofiles *(*ir_isofiles_storage*)*, 22
ir_show_problems *(*ir_get_problems*)*, 21
ir_start_aggregator, 34
ir_start_aggregator(), *5*, *35*
ir_storage, 36

isoreader2 *(*isoreader2-package*)*, 2
isoreader2-package, 2

numeric_version, *13*

options(), *24*

palette.colors(), *27*, *30*, *32*

readr::read_rds(), *22*
readr::write_rds(), *22*

tidyselect, *15*, *18*, *26*, *29*, *31*