Package: isoexplorer (via r-universe)

July 22, 2026

Title GUI Components to Explore Stable Isotope Data Files

Version 0.4.1

Description Provides graphical user interface components to explore stable isotope data files using the 'isoreader2' package, including browsing, visualizing, and exporting isotope data.

License AGPL (>= 3)

Encoding UTF-8

Depends R (>= 4.5.0)

Imports cli (>= 3.6.0), rlang (>= 1.1.0), tibble, dplyr, purrr, stringr, htmltools, shiny, bslib, shinyjs, shinyAce, shinycssloaders, rlog, DT, ggplot2, isoreader2 (>= 0.6.0), stats, callr, httpuv, htmlwidgets, tidyr, tidyselect, tools, utils, withr

Suggests testthat (>= 3.0.0), pkgload

Config/testthat/edition 3

URL https://github.com/isoverse/isoexplorer

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

Language en-US

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 https://cran.r-universe.dev

Date/Publication 2026-07-22 09:43:30 UTC

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

RemoteRef HEAD

RemoteSha b39b11c472365eb094417071d2a2b9e2e4cc6ead

Contents

`ie_cf_plot` *Continuous flow plot module*

Description

A plot view for continuous flow trace data, wired to a `ie_file_server()`: it plots the selection-filtered aggregated `traces` with `isoreader2::ir_plot_continuous_flow()`, with unit, species/mass, legend, zoom and PDF-download controls. Pair `ie_cf_plot_ui()` and `ie_cf_plot_server()` on one `id`.

Usage

```
ie_cf_plot_ui(id)

ie_cf_plot_server(id, file)
```

Arguments

`id`	the module id (must match the paired `ie_metadata_ui()`)
`file`	the `ie_file_server()` handle

Value

`ie_cf_plot_ui()` returns a UI element; `ie_cf_plot_server()` returns a list with a `get_code` generator for the code server (see `ie_code_server()`)

See Also

`ie_file_server()`, `ie_cf_metadata_server()`

Examples

```
if (interactive()) {
  library(shiny)
  iso <-
    isoreader2::ir_examples_folder() |>
    isoreader2::ir_find_isofiles() |>
    isoreader2::ir_read_isofiles()

  ui <- ie_type_explorer_ui("meta", ie_cf_plot_ui("cf"))
  server <- function(input, output, session) {
    file <- ie_file_server("files", get_isofiles = reactive(iso))
    ie_cf_metadata_server("meta", file)
    ie_cf_plot_server("cf", file)
  }
  shinyApp(ui, server)
}
```

`ie_code_server` *Code-generation server*

Description

A central module that assembles "idealized example code" for an isoexplorer app and shows it in a viewer. It is instantiated once at the top level; every other module that can contribute code returns a `get_code()` generator, and the top-level wiring **registers** each one here with `register()` under an id, a heading, and (optionally) the id it `depends_on`.

Usage

```
ie_code_server(id, get_active_group = reactive(NULL))

ie_code_ui(id)
```

Arguments

`id`	the module id (namespace); pair with `ie_code_ui()` on the same id
`get_active_group`	a reactive returning the active group id; only registrations in this group (plus group-less ones) are shown. The default `reactive(NULL)` shows everything.

Details

The registrations form a dependency tree. When the user clicks the navbar **Show code** button (`ie_code_ui()`), the registered generators are walked depth-first and assembled into one document: the tree depth sets the heading level (# for a root such as a read step, ## for a step depending on a root, ### for one depending on that, ...), and each step's `output` variable is threaded in as the `input_var` of the steps that depend on it. The document is shown in a read-only `shinyAce::aceEditor()`

with a clickable headings tree (jump-to-section), a toggle between plain code and a Quarto view, and a `.qmd` download.

Each registration may carry a `group`; when `get_active_group()` returns a non-empty value only registrations in that group (plus group-less ones) are shown – the full multi-tab app uses this to show just the active measurement type's code.

Value

the code handle: a list with `register(code_id, heading, get_code, depends_on = NULL, group = NULL)` (each `depends_on` is a single id – the tree is single-parent) and `build_document(quarto = FALSE)` (assemble + return `list(script, headings)`; exposed mainly for testing).

Functions

- `ie_code_ui()`: the navbar **Show code** button (place in the navbar, e.g. `bslib::nav_item(ie_code_ui("code"))`) plus the one-time JS used to jump the editor to a heading. Pair with `ie_code_server()` on the same `id`.

The `get_code()` contract

a registered generator is `function(input_var = NULL)` returning `list(code = <string>, output = <string or NULL>)`. It is called during assembly inside a reactive context, so it reflects the module's *current* state; `input_var` is the output variable of the module it depends on (`NULL` for a root), and `output` is the variable this snippet binds (or `NULL` for a terminal node such as a plot).

Examples

```
if (interactive()) {
  library(shiny)
  # ie_run_app() instantiates the code server for you; here it is wired
  # directly. Register each module's get_code generator into the dependency
  # tree; the "Show code" button (ie_code_ui()) assembles and displays it.
  ui <- bslib::page_fillable(ie_code_ui("code"))
  server <- function(input, output, session) {
    code <- ie_code_server("code")
    code$register("read", "Read data files", get_code = function(input_var = NULL) {
      list(
        code = 'iso <- ir_find_scans("data") |> ir_read_isofiles()',
        output = "iso"
      )
    })
    code$register("agg", "Aggregate data files", depends_on = "read",
      get_code = function(input_var = NULL) {
        list(
          code = sprintf("scans <- %s |> ir_aggregate_isofiles()", input_var),
          output = "scans"
        )
      })
  }
  shinyApp(ui, server)
}
```

`ie_create_isofiles_server` *Create an isofiles server app for the isoexplorer GUI*

Description

Builds the full isoexplorer Shiny app – one navbar tab per measurement type (continuous flow / dual inlet / scans), each with a file-selector sidebar and plot – and returns it as a `shiny::shinyApp()` object to run or deploy. Unlike the focused `ie_explore_continuous_flow()` explorers, this app does NOT take an `ir_isofiles` object – data arrives at runtime via the navbar **Upload** button, any watched `monitoring_folders`, and/or the **Load examples** button (a "get started" prompt is shown until something is loaded). Loaded examples are selected automatically; uploaded files are selected only when the upload modal's "Select the uploaded files" box is checked.

Usage

```
ie_create_isofiles_server(
  timezone = Sys.timezone(),
  options = list(),
  uiPattern = "/",
  enableBookmarking = "url",
  default_theme = app_themes(),
  upload_folder = NULL,
  monitoring_folders = NULL,
  examples_folder = "examples",
  temporary_storage = FALSE,
  max_upload_size = NULL,
  log_level = "TRACE"
)
```

Arguments

`timezone`	the timezone to use for datetime display
`options`	Named options that should be passed to the `runApp` call (these can be any of the following: "port", "launch.browser", "host", "quiet", "display.mode" and "test.mode"). You can also specify `width` and `height` parameters which provide a hint to the embedding environment about the ideal height/width for the app.
`uiPattern`	A regular expression that will be applied to each `GET` request to determine whether the `ui` should be used to handle the request. Note that the entire request path must match the regular expression in order for the match to be considered successful.
`enableBookmarking`	Can be one of `"url"`, `"server"`, or `"disable"`. The default value, `NULL`, will respect the setting from any previous calls to `enableBookmarking()`. See `enableBookmarking()` for more information on bookmarking your app.

`default_theme` the default bslib Bootstrap 5 theme preset

`upload_folder` upload directory for the navbar upload button; `NULL` (the default) means no upload button, a path enables it; see `ie_file_server()`

`monitoring_folders` folders to watch for new isofiles, read and added automatically (`NULL` = off); see `ie_file_server()`

`examples_folder` directory the "Load examples" navbar button copies the isoreader2 bundled example files into and loads; `"examples"` by default (`NULL` hides the button)

`temporary_storage` if `TRUE`, the upload dialog notes that uploaded files are stored only for the duration of the session (informational; default `FALSE`)

`max_upload_size` maximum per-file upload size in MB (sets the `shiny.maxRequestSize` option); `NULL` (the default) keeps Shiny's ~5 MB default. Raw isofiles are often larger, so raise this when allowing uploads.

`log_level` how verbosely the app logs; see `ie_run_app()`. Defaults to `"TRACE"` (log everything) for this server app, unlike the focused explorers which default to `"WARN"`.

Details

Because it returns the app object unrun, launch it yourself (print it, or wrap it in `shiny::runApp()`) or hand it to a deployment tool (e.g. shinyapps.io / ShinyProxy).

Value

a `shiny::shinyApp()` object (unrun)

Examples

```
if (interactive()) {
  # build the full multi-tab server app and run it; load data via the navbar
  # "Load examples" / "Upload" buttons at runtime
  ie_create_isofiles_server() |> shiny::runApp()

  # enable uploads (raise the per-file cap for large raw isofiles)
  app <- ie_create_isofiles_server(
    upload_folder = "uploads",
    max_upload_size = 200
  )
  shiny::runApp(app)
}
```

ie_di_plot *Dual inlet plot module*

Description

A plot view for dual inlet cycle data, wired to a `ie_file_server()`: it plots the selection-filtered aggregated `cycles` with `isoreader2::ir_plot_dual_inlet()`, with unit, species/mass, legend, zoom and PDF-download controls. Pair `ie_di_plot_ui()` and `ie_di_plot_server()` on one `id`.

Usage

```
ie_di_plot_ui(id)

ie_di_plot_server(id, file)
```

Arguments

`id`	the module id (must match the paired `ie_metadata_ui()`)
`file`	the `ie_file_server()` handle

Value

`ie_di_plot_ui()` returns a UI element; `ie_di_plot_server()` returns a list with a `get_code` generator for the code server (see `ie_code_server()`)

See Also

`ie_file_server()`, `ie_di_metadata_server()`

Examples

```
if (interactive()) {
  library(shiny)
  iso <-
    isoreader2::ir_examples_folder() |>
    isoreader2::ir_find_isofiles() |>
    isoreader2::ir_read_isofiles()

  ui <- ie_type_explorer_ui("meta", ie_di_plot_ui("di"))
  server <- function(input, output, session) {
    file <- ie_file_server("files", get_isofiles = reactive(iso))
    ie_di_metadata_server("meta", file)
    ie_di_plot_server("di", file)
  }
  shinyApp(ui, server)
}
```

`ie_explore_continuous_flow` *Explore an ir_isofiles object by measurement type*

Description

Focused apps for exploring an already-read `ir_isofiles` object one measurement type at a time: `ie_explore_continuous_flow()` / `ie_explore_dual_inlet()` / `ie_explore_scans()` each show that type's file-selector sidebar + plot, and `ie_explore_metadata()` shows just the selector table. The generated example code (navbar **Show code**) refers to the object by its `variable_name`. These take a fixed object only – for upload / folder monitoring / load-examples use `ie_create_isofiles_server()`.

Usage

```
ie_explore_continuous_flow(
  isofiles,
  timezone = Sys.timezone(),
  options = list(),
  uiPattern = "/",
  enableBookmarking = "url",
  default_theme = app_themes(),
  initial_selection = FALSE,
  variable_name = NULL,
  detached = TRUE,
  launch = TRUE,
  log_level = "WARN"
)

ie_explore_dual_inlet(
  isofiles,
  timezone = Sys.timezone(),
  options = list(),
  uiPattern = "/",
  enableBookmarking = "url",
  default_theme = app_themes(),
  initial_selection = FALSE,
  variable_name = NULL,
  detached = TRUE,
  launch = TRUE,
  log_level = "WARN"
)

ie_explore_scans(
  isofiles,
  timezone = Sys.timezone(),
  options = list(),
  uiPattern = "/",
```

```
  enableBookmarking = "url",
  default_theme = app_themes(),
  initial_selection = FALSE,
  variable_name = NULL,
  detached = TRUE,
  launch = TRUE,
  log_level = "WARN"
)

ie_explore_metadata(
  isofiles,
  timezone = Sys.timezone(),
  options = list(),
  uiPattern = "/",
  enableBookmarking = "url",
  default_theme = app_themes(),
  initial_selection = FALSE,
  variable_name = NULL,
  detached = TRUE,
  launch = TRUE,
  log_level = "WARN"
)
```

Arguments

isofiles	the ir_isofiles object to explore (required)
timezone	timezone for datetime display
options	Named options that should be passed to the runApp call (these can be any of the following: "port", "launch.browser", "host", "quiet", "display.mode" and "test.mode"). You can also specify width and height parameters which provide a hint to the embedding environment about the ideal height/width for the app.
uiPattern	A regular expression that will be applied to each GET request to determine whether the ui should be used to handle the request. Note that the entire request path must match the regular expression in order for the match to be considered successful.
enableBookmarking	Can be one of "url", "server", or "disable". The default value, NULL, will respect the setting from any previous calls to enableBookmarking(). See enableBookmarking() for more information on bookmarking your app.
default_theme	default bslib Bootstrap 5 theme preset
initial_selection	what is selected on load, as a dplyr::filter() expression on the aggregated metadata: FALSE (the default) selects nothing, TRUE selects everything, and any other expression (e.g. grepl("std", file_name)) selects the matching files/analyses. See ie_file_server(). In detached mode the expression is re-evaluated in the separate process, so it must be self-contained (it cannot reference variables from your session).

variable_name	the name used for the object in the generated example code. Defaults to the deparsed expression you passed (so my_iso |> ie_explore_scans() uses "my_iso"); set it explicitly to override.
detached	if TRUE, the app is launched in a separate R process (via callr::r_bg(), the app handed over in a temporary .rds) and opened in your default browser, leaving the calling session free; closing the browser tab stops the app and the process (which is also killed if the calling session exits). Default FALSE.
launch	only relevant when detached = FALSE: if TRUE (the default) the app is run in the current session (blocking) with shiny::runApp(); if FALSE the shiny::shinyApp() object is returned unrun (for server deployment or manual launching). Ignored when detached = TRUE (a detached app is always run).
log_level	how verbosely the app logs; see ie_run_app() for the available levels. Defaults to "WARN" for the focused explorers.

Details

By default the app runs **detached** in a separate R process (see detached), so the calling session is not blocked, and it refuses to launch while a document is being rendered (knitr / Quarto), showing a message to run it interactively. These are thin wrappers over ie_run_app(), which manages detached / launch.

Value

the value of ie_run_app() for the chosen detached / launch: the callr::r_bg() process (detached), the shiny::runApp() result (in-session launch), or the shiny::shinyApp() object (launch = FALSE).

Functions

- ie_explore_dual_inlet(): focused app for the dual inlet plot.
- ie_explore_scans(): focused app for the scans plot.
- ie_explore_metadata(): focused app showing just the scans metadata selector table (handy for browsing/testing the selector).

Examples

```
if (interactive()) {
  # read the bundled isoreader2 examples (a mixed set of all types); each
  # explorer filters the object to its own measurement type
  iso <-
    isoreader2::ir_examples_folder() |>
    isoreader2::ir_find_isofiles() |>
    isoreader2::ir_read_isofiles()

  # each focused explorer opens the object in a detached browser app
  ie_explore_continuous_flow(iso)
  ie_explore_dual_inlet(iso)
  ie_explore_scans(iso)
  ie_explore_metadata(iso) # just the selector table
```

```
  # preselect some files and run blocking in the current session instead
  ie_explore_continuous_flow(
    iso,
    initial_selection = grepl("gc", file_name),
    detached = FALSE
  )
}
```

`ie_file_server` *Central file-management module*

Description

The single source of truth for the isofiles in an isoexplorer app, and the hub every other module talks to. It maintains a *running* set of read isofiles (seeded from `get_isofiles()`, grown by uploads and watched folders), splits it into the three measurement types (scans / continuous flow / dual inlet) with `isoreader2::ir_filter_for_scans()` and friends, owns the shared intensity-units selection and the per-type file selection, and exposes the per-type metadata (for the `ie_metadata_server()` selector tables) and the selection-filtered aggregated data (for the `*_plot_server()` modules).

Usage

```
ie_file_server(
  id,
  get_isofiles,
  initial_selection = TRUE,
  upload_folder = NULL,
  monitoring_folders = NULL,
  examples_folder = NULL,
  temporary_storage = FALSE
)

ie_file_ui(id)
```

Arguments

`id`	the module id (namespace)
`get_isofiles`	a reactive returning an `ir_isofiles` to seed the app with (already read); `reactive(NULL)` is fine when files only arrive via upload / monitoring
`initial_selection`	what is selected, per type, before any selector pushes a selection. An expression evaluated as a `dplyr::filter()` on the type's aggregated metadata tibble: `TRUE` (default) selects everything, `FALSE` selects nothing, and any other expression selects the matching rows (e.g. `grepl("std", file_name)`). Captured via tidy evaluation, so it may reference variables from the calling environment. Passed pre-quoted (a quosure) by the `ie_explore_*()` / `ie_run_app()` wrappers.

`upload_folder` directory where uploaded files are stored (created on demand); `NULL` (the default) means no upload button – set a directory path to enable uploads.

`monitoring_folders`
character vector of folders to watch; isofiles found there with `isoreader2::ir_find_isofiles()` are read and added automatically. `NULL` (default) disables monitoring.

`examples_folder`
directory the "Load examples" navbar button copies the isoreader2 bundled example files into (and then loads). `NULL` (the default) means no examples button.

`temporary_storage`
if `TRUE`, the upload dialog states that uploaded files are stored only for the duration of the session. Informational only – it does not change how or where files are stored (default `FALSE`).

Details

Selector tables read `get_<type>_metadata()` and push their selection via `set_selected_<type>()`; plot modules read `get_aggregated_<type>_data()` and drive the shared units via `get_units()` / `set_units()`.

Dynamic files. New files (uploaded, or appearing in `monitoring_folders`) are read with `isoreader2::ir_read_isofiles` and appended; aggregation is incremental (only new files are read/aggregated, then combined with `c()`), so already-read files are never re-read.

Upload. When `upload_folder` is set the module owns a navbar upload button (the `ie_file_ui()` placeholder) that opens a modal to upload multiple files or whole folders bundled as `.zip` archives (the picker allows `.zip`, the file types isoreader2 reads, and `.json` – which covers their `.<type>.json` serializations such as `foo.cf.json`). Uploaded files are stored in `upload_folder` (archives unpacked), and **only the just-uploaded files are read** – files already present in `upload_folder` when the app started are left untouched. An "Auto-select the newly uploaded files" checkbox (off by default) exclusively selects the new files in the relevant type's table and, in the full multi-tab app, switches to that type's tab.

Monitoring. `monitoring_folders` are polled; any isofiles found there with `isoreader2::ir_find_isofiles()` (including files already present at startup) are read and added.

Getting started. When `examples_folder` and/or `upload_folder` is set but the app is launched without data (empty `get_isofiles()`), a prompt is shown once inviting the user to load the examples and/or upload their own files; an app launched with data never sees it.

Value

The "file handle": a list of reactive accessors / setters. For each `<type>` in `scans` / `cf` / `di`: `get_units()`/`set_units(units)` (shared intensity units, default "mV"); `get_<type>_metadata()`; `set_selected_<type>(rows)`; `get_<type>_selection()`; `get_aggregated_<type>_data()`; plus `get_<type>_select_signal()` (file paths a selector should select, fired by upload auto-select) and `get_active_type()` (the type whose tab to activate after an auto-select).

Functions

- `ie_file_ui()`: the navbar placeholders for the "Load examples" and "Upload" buttons (each rendered only when the server's `examples_folder` / `upload_folder` is set). Pair with `ie_file_server()` on the same `id`.

Examples

```
if (interactive()) {
  library(shiny)
  # read the bundled isoreader2 examples for some scans data
  iso <-
    isoreader2::ir_examples_folder() |>
    isoreader2::ir_find_isofiles() |>
    isoreader2::ir_read_isofiles()

  ui <- bslib::page_fillable(
    ie_type_explorer_ui("meta", ie_scans_plot_ui("scan"))
  )
  server <- function(input, output, session) {
    # the hub: every other module reads/writes through this handle
    file <- ie_file_server("files", get_isofiles = reactive(iso))
    ie_scans_metadata_server("meta", file) # selection -> file server
    ie_scans_plot_server("scan", file)     # file server -> plot
  }
  shinyApp(ui, server)
}
```

`ie_metadata_server` *Metadata selector module*

Description

Server for a file/analysis selector table (pair with `ie_metadata_ui()`). It shows a row-grouped table over an already-aggregated metadata tibble and pushes the current selection out via `set_selected`; double-clicking a row selects just that analysis, double-clicking a file's group header selects all of its analyses. Most users want the typed wrappers `ie_scans_metadata_server()` / `ie_cf_metadata_server()` / `ie_di_metadata_server()`, which bind the right `ie_file_server()` accessors; `ie_metadata_server()` itself is the generic version for wiring a custom metadata source.

Usage

```
ie_metadata_server(
  id,
  get_metadata,
  set_selected = NULL,
  get_selection = NULL,
  get_select_signal = NULL
)
```

Arguments

`id`	the module id (must match the paired `ie_metadata_ui()`)
`get_metadata`	a reactive returning the metadata tibble to browse (one row per analysis, with `uidx` + `analysis` columns)

set_selected — optional function(rows) called with the selected metadata rows whenever the selection changes (e.g. file$set_selected_scans)

get_selection — optional reactive of the current selection, used to reflect it in the table on load (e.g. file$get_scans_selection)

get_select_signal — optional reactive carrying file paths the table should exclusively select (e.g. file$get_scans_select_signal, fired by upload auto-select); applied once the table contains those files

Value

a list with the underlying selector-table handle plus the get_selected_row_id / get_selected_metadata reactives

Examples

```
if (interactive()) {
  library(shiny)
  iso <-
    isoreader2::ir_examples_folder() |>
    isoreader2::ir_find_isofiles() |>
    isoreader2::ir_read_isofiles()

  ui <- ie_metadata_ui("meta")
  server <- function(input, output, session) {
    file <- ie_file_server("files", get_isofiles = reactive(iso))
    # the generic selector wired to any aggregated-metadata accessors; most
    # users want the typed wrappers ie_scans_metadata_server() / _cf_ / _di_
    ie_metadata_server(
      "meta",
      get_metadata = file$get_scans_metadata,
      set_selected = file$set_selected_scans,
      get_selection = file$get_scans_selection
    )
  }
  shinyApp(ui, server)
}
```

---

ie_metadata_ui *Metadata selector table UI*

Description

The UI for a ie_metadata_server() file/analysis selector: a fullscreen-capable card with a toolbar (select-all / deselect on the left, column / search controls on the right) above a table that fills the rest of the card. Pair it with one of the *_metadata_server() functions using the same id.

Usage

```
ie_metadata_ui(id)
```

Arguments

id	the module id (must match the paired *_metadata_server())

Value

a bslib::card() UI element

See Also

ie_scans_metadata_server(), ie_file_server()

Examples

```
# pair with a *_metadata_server() on the same id inside a shiny app
ie_metadata_ui("meta")
```

---

ie_run_app *Assemble and launch a custom isoexplorer app*

Description

Wraps your own module composition in the isoexplorer navbar shell (theme picker, dark-mode toggle, about popup) and, by default, runs it. Build your layout from the package's modules, instantiate the ie_file_server() and wire the modules in setup_modules. Supply EITHER main (a single content area) OR nav_panels (a centered navbar tabset).

Usage

```
ie_run_app(
  isofiles,
  main = NULL,
  setup_modules,
  timezone = Sys.timezone(),
  default_theme = app_themes(),
  nav_panels = NULL,
  selected = NULL,
  initial_selection = TRUE,
  upload_folder = NULL,
  monitoring_folders = NULL,
  examples_folder = NULL,
  temporary_storage = FALSE,
  max_upload_size = NULL,
  options = list(),
  uiPattern = "/",
  enableBookmarking = "url",
  detached = FALSE,
  launch = TRUE,
```

```
  stop_on_close = detached,
  log_level = c("WARN", "INFO", "DEBUG", "TRACE", "ERROR", "FATAL")
)
```

Arguments

isofiles	the ir_isofiles to explore (handed to the ie_file_server())
main	a single UI content area (use this OR nav_panels)
setup_modules	function(file, code) that wires the app's server modules, given the ie_file_server() handle and the ie_code_server() handle (register each module's get_code with code$register()). A one-argument function(file) is still accepted (the code server is then not populated).
timezone	timezone for datetime display
default_theme	default bslib Bootstrap 5 theme preset
nav_panels	a list of bslib::nav_panel()s shown as a centered navbar tabset (use this OR main)
selected	the value/title of the nav_panels tab to open initially (NULL = the first tab)
initial_selection	initial file selection, see ie_file_server()
upload_folder	upload directory for the navbar upload button; NULL (the default) means no upload button, see ie_file_server()
monitoring_folders	folders to watch for new isofiles (NULL = off), see ie_file_server()
examples_folder	directory for the "Load examples" navbar button (NULL = off), see ie_file_server()
temporary_storage	note in the upload dialog that uploads are session-only (informational; default FALSE), see ie_file_server()
max_upload_size	maximum per-file upload size in MB; sets the shiny.maxRequestSize option for the running app. NULL (the default) leaves Shiny's ~5 MB default (or a value you set yourself) untouched. Raw isofiles are often larger than 5 MB, so raise this when enabling uploads.
options	Named options that should be passed to the runApp call (these can be any of the following: "port", "launch.browser", "host", "quiet", "display.mode" and "test.mode"). You can also specify width and height parameters which provide a hint to the embedding environment about the ideal height/width for the app.
uiPattern	A regular expression that will be applied to each GET request to determine whether the ui should be used to handle the request. Note that the entire request path must match the regular expression in order for the match to be considered successful.
enableBookmarking	Can be one of "url", "server", or "disable". The default value, NULL, will respect the setting from any previous calls to enableBookmarking(). See enableBookmarking() for more information on bookmarking your app.

detached	if TRUE, the app is launched in a separate R process (via callr::r_bg(), the app handed over in a temporary .rds) and opened in your default browser, leaving the calling session free; closing the browser tab stops the app and the process (which is also killed if the calling session exits). Default FALSE.
launch	only relevant when detached = FALSE: if TRUE (the default) the app is run in the current session (blocking) with shiny::runApp(); if FALSE the shiny::shinyApp() object is returned unrun (for server deployment or manual launching). Ignored when detached = TRUE (a detached app is always run).
stop_on_close	if TRUE, the app stops itself when the browser disconnects (session$onSessionEnded()); set automatically for the detached process so it exits when the tab is closed. Defaults to the same as detached.
log_level	how verbosely the app logs, one of "WARN" (the default – only warnings and errors), "INFO", "DEBUG", "TRACE" (everything), "ERROR", or "FATAL". Sets the LOG_LEVEL environment variable that rlog reads (also applied in the detached process).

Details

The detached / launch arguments control how it runs: detached = TRUE launches it in a separate R process (leaving this session free); detached = FALSE, launch = TRUE (the default) runs it in the current session (blocking) with shiny::runApp(); detached = FALSE, launch = FALSE returns the shiny::shinyApp() object unrun (for deployment or manual launching). The focused explorers (ie_explore_continuous_flow() etc.) and ie_create_isofiles_server() are wrappers that call this with the appropriate detached / launch.

Value

When detached = TRUE, the callr::r_bg() process (invisibly, with the app URL attached as an attribute); when detached = FALSE and launch = TRUE, the result of shiny::runApp() (after the app stops); when detached = FALSE and launch = FALSE, the shiny::shinyApp() object.

Examples

```
if (interactive()) {
  # read the bundled isoreader2 examples (a mixed set of all types)
  iso <-
    isoreader2::ir_examples_folder() |>
    isoreader2::ir_find_isofiles() |>
    isoreader2::ir_read_isofiles()

  # a minimal custom app: a scans selector next to a scans plot
  ie_run_app(
    isofiles = iso,
    main = ie_type_explorer_ui("meta", ie_scans_plot_ui("scan")),
    setup_modules = function(file, code) {
      ie_scans_metadata_server("meta", file)
      ie_scans_plot_server("scan", file)
    }
  )
}
```

ie_scans_plot *Scans plot module*

Description

A plot view for scan data, wired to a ie_file_server(): it plots the selection-filtered aggregated scans with isoreader2::ir_plot_scans(), with unit, species/mass, legend, zoom and PDF-download controls plus a scan-type popover (scans are plotted one type at a time). Pair ie_scans_plot_ui() and ie_scans_plot_server() on one id.

Usage

```
ie_scans_plot_ui(id)

ie_scans_plot_server(id, file)
```

Arguments

id	the module id (must match the paired ie_metadata_ui())
file	the ie_file_server() handle

Value

ie_scans_plot_ui() returns a UI element; ie_scans_plot_server() returns a list with a get_code generator for the code server (see ie_code_server())

See Also

ie_file_server(), ie_scans_metadata_server()

Examples

```
if (interactive()) {
  library(shiny)
  iso <-
    isoreader2::ir_examples_folder() |>
    isoreader2::ir_find_isofiles() |>
    isoreader2::ir_read_isofiles()

  ui <- ie_type_explorer_ui("meta", ie_scans_plot_ui("scan"))
  server <- function(input, output, session) {
    file <- ie_file_server("files", get_isofiles = reactive(iso))
    ie_scans_metadata_server("meta", file)
    ie_scans_plot_server("scan", file) # reads the selection-filtered data
  }
  shinyApp(ui, server)
}
```

ie_type_explorer_ui *File-selector + plot layout for one measurement type*

Description

Convenience UI combining a left file-selector sidebar (a `ie_metadata_ui()`) with a plot to its right – the building block each focused explorer and each tab of `ie_create_isofiles_server()` uses. Wire the matching `*_metadata_server()` (on `meta_id`) and `*_plot_server()` in your server function.

Usage

```
ie_type_explorer_ui(meta_id, plot_ui)
```

Arguments

`meta_id`	the id for the `ie_metadata_ui()` / `*_metadata_server()` pair
`plot_ui`	a plot module UI element, e.g. `ie_scans_plot_ui("scan")`

Value

a `bslib::layout_sidebar()` UI element

See Also

`ie_file_server()`, `ie_scans_plot_ui()`, `ie_scans_metadata_server()`

Examples

```
# a file-selector sidebar next to a scans plot (place inside a shiny UI)
ie_type_explorer_ui("meta", ie_scans_plot_ui("scan"))
```

ie_typed_metadata_servers *Typed metadata selectors*

Description

Selector-table servers wired to a `ie_file_server()` for one measurement type: they read that type's metadata and push the selection back into the file server. Pair each with a `ie_metadata_ui()` using the same `id`.

Usage

```
ie_scans_metadata_server(id, file)

ie_cf_metadata_server(id, file)

ie_di_metadata_server(id, file)
```

Arguments

id	the module id (must match the paired ie_metadata_ui())
file	the ie_file_server() handle

Value

a ie_metadata_server() handle

See Also

ie_file_server(), ie_metadata_ui()

Examples

```
if (interactive()) {
  library(shiny)
  iso <-
    isoreader2::ir_examples_folder() |>
    isoreader2::ir_find_isofiles() |>
    isoreader2::ir_read_isofiles()

  ui <- ie_metadata_ui("meta")
  server <- function(input, output, session) {
    file <- ie_file_server("files", get_isofiles = reactive(iso))
    # pushes the table's selection into the file server for this type
    ie_scans_metadata_server("meta", file)
  }
  shinyApp(ui, server)
}
```

Index