Package: flow (via r-universe)

September 14, 2024

Title View and Browse Code Using Flow Diagrams

Version 0.2.0.9000

Description Visualize as flow diagrams the logic of functions, expressions or scripts in a static way or when running a call, visualize the dependencies between functions or between modules in a shiny app, and more.

License MIT + file LICENSE

URL https://github.com/moodymudskipper/flow, https://moodymudskipper.github.io/flow/

BugReports https://github.com/moodymudskipper/flow/issues

Encoding UTF-8

Imports nomnoml, utils, htmlwidgets, rstudioapi, webshot, styler, methods, here, lifecycle

Suggests testthat (>= 3.0.0), covr, knitr, rmarkdown, esquisse, tidyselect, purrr

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Config/testthat/edition 3

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

RemoteUrl https://github.com/moodymudskipper/flow

RemoteRef HEAD

RemoteSha 1645fe797764c3189a5800a037e0854bc8fabfeb

Contents

`flow_debug` *Debug With Flow Diagrams*

Description

These functions are named after the base functions `debug()` and `undebug()`. `flow_debug()` will call `flow_run()`, with the same additional arguments, on all the following calls to `f()` until `flow_undebug()` is called.

Usage

```
flow_debug(
  f,
  prefix = NULL,
  code = TRUE,
  narrow = FALSE,
  truncate = NULL,
  swap = TRUE,
  out = NULL,
  browse = FALSE
)

flow_undebug(f)
```

Arguments

`f`	function to debug
`prefix`	prefix to use for special comments in our code used as block headers, must start with `"#"`, several prefixes can be provided
`code`	Whether to display the code in code blocks or only the header, to be more compact, if `NA`, the code will be displayed only if no header is defined by special comments
`narrow`	`TRUE` makes sure the diagram stays centered on one column (they'll be longer but won't shift to the right)
`truncate`	maximum number of characters to be printed per line

swap	whether to change `var <- if(cond) expr` into `if(cond) var <- expr` so the diagram displays better
out	a path to save the diagram to. Special values "html", "htm", "png", "pdf", "jpg" and "jpeg" can be used to export the object to a temp file of the relevant format and open it, if a regular path is used the format will be guessed from the extension.
browse	whether to debug step by step (block by block), can also be a vector of block ids, in this case `browser()` calls will be inserted at the start of these blocks

Details

By default, unlike `debug()`, `flow_debug()` doesn't trigger a debugger but only draw diagrams, this is consistent with `flow_run()`'s defaults. To browse through the code, use the `browse` argument.

Value

These functions return `NULL` invisibly (called for side effects)

---

flow_doc *Draw Flow Diagrams for an Entire Package*

Description

Draw Flow Diagrams for an Entire Package

Usage

```
flow_doc(
  pkg = NULL,
  prefix = NULL,
  code = TRUE,
  narrow = FALSE,
  truncate = NULL,
  swap = TRUE,
  out = NULL,
  engine = c("nomnoml", "plantuml")
)
```

Arguments

pkg	package name as a string, or `NULL` to signify currently developed package.
prefix	prefix to use for special comments in our code used as block headers, must start with `"#"`, several prefixes can be provided
code	Whether to display the code in code blocks or only the header, to be more compact, if `NA`, the code will be displayed only if no header is defined by special comments

narrow	TRUE makes sure the diagram stays centered on one column (they'll be longer but won't shift to the right)
truncate	maximum number of characters to be printed per line
swap	whether to change var <- if(cond) expr into if(cond) var <- expr so the diagram displays better
out	path to html output, if left NULL a temp *html* file will be created and opened
engine	either "nomnoml" (default) or "plantuml" (experimental, brittle mostly for reasons out of our control), if the latter, arguments prefix, narrow, and code are ignored

Value

Returns NULL invisibly (called for side effects).

`flow_draw` *Draw Diagram From Debugger*

Description

`flow_draw()` should only be used in the debugger triggered by a call to `flow_run()`, or following a call to `flow_debug()`. `d` is an active binding to `flow_draw()`, it means you can just type `d` (without parentheses) instead of `flow_draw()`.

Usage

```
flow_draw()

d
```

Details

`d` was designed to look like the other shortcuts detailed in `?browser`, such as `f`, `c` etc... It differs however in that it can be overridden. For instance if the function uses a variable `d` or that a parent environment contains a variable `d`, `flow::d` won't be found. In that case you will have to use `flow_draw()`.

If `d` or `flow_draw()` are called outside of the debugger they will return NULL silently.

Value

Returns NULL invisibly (called for side effects)

flow_embed *Embed chart in roxygen doc*

Description

Include a call `r_flow::flow_embed(...)` in your doc and a diagram will be included.

Usage

```
flow_embed(call, name, width = 1, alt = name)
```

Arguments

call	A call to a flow function, prefixed with flow::
name	A name for the png file that will be created under 'man/figures', without extension.
width	width, relative if < 1, pixels otherwise
alt	alt text

Details

- As with images in general the image might not be visible when viewing temp doc with the devtools workflow.
- Don't forget to add **flow** to Suggests in your DESCRIPTION file.
- We don't monitor files created under 'man/figures', so if you remove a diagram from the doc make sure to also remove it from the folder.
- We also don't overwrite created files, so we don't slow down the documentation process, so if you want to print a different diagram for the same name remove the file first.

Value

Called for side effects, should only be used in roxygen doc

flow_test *Build Report From Tests*

Description

Build a markdown report from test scripts, showing the paths taken in tested functions, and where they fail if they do. See also the vignette *"Build reports to document functions and unit tests"*.

Usage

```
flow_test(
  prefix = NULL,
  code = TRUE,
  narrow = FALSE,
  truncate = NULL,
  swap = TRUE,
  out = NULL,
  failed_only = FALSE
)
```

Arguments

prefix	prefix to use for special comments in our code used as block headers, must start with "#", several prefixes can be provided
code	Whether to display the code in code blocks or only the header, to be more compact, if NA, the code will be displayed only if no header is defined by special comments
narrow	TRUE makes sure the diagram stays centered on one column (they'll be longer but won't shift to the right)
truncate	maximum number of characters to be printed per line
swap	whether to change var <- if(cond) expr into if(cond) var <- expr so the diagram displays better
out	path to html output, if left NULL a temp *html* file will be created and opened.
failed_only	whether to restrict the report to failing tests only

Value

Returns NULL invisibly (called for side effects)

---

flow_view *View function as flow chart*

Description

- flow_view() shows the code of a function as a flow diagram
- flow_run() runs a call and draws the logical path taken by the code.
- flow_compare_runs() shows on the same diagrams 2 calls to the same functions, code blocks that are only touched by the ref call are colored green, code blocks that are only touched by the x call are colored orange.

Usage

```
flow_view(
  x,
  prefix = NULL,
  code = TRUE,
  narrow = FALSE,
  truncate = NULL,
  nested_fun = NULL,
  swap = TRUE,
  out = NULL,
  engine = c("nomnoml", "plantuml")
)

flow_run(
  x,
  prefix = NULL,
  code = TRUE,
  narrow = FALSE,
  truncate = NULL,
  swap = TRUE,
  out = NULL,
  browse = FALSE
)

flow_compare_runs(
  x,
  ref,
  prefix = NULL,
  code = TRUE,
  narrow = FALSE,
  truncate = NULL,
  swap = TRUE,
  out = NULL
)
```

Arguments

x	a call, a function, or a path to a script
prefix	prefix to use for special comments in our code used as block headers, must start with "#", several prefixes can be provided
code	Whether to display the code in code blocks or only the header, to be more compact, if NA, the code will be displayed only if no header is defined by special comments
narrow	TRUE makes sure the diagram stays centered on one column (they'll be longer but won't shift to the right)
truncate	maximum number of characters to be printed per line

nested_fun	if not NULL, the index or name of the function definition found in x that we wish to inspect
swap	whether to change var <- if(cond) expr into if(cond) var <- expr so the diagram displays better
out	a path to save the diagram to. Special values "html", "htm", "png", "pdf", "jpg" and "jpeg" can be used to export the object to a temp file of the relevant format and open it, if a regular path is used the format will be guessed from the extension.
engine	either "nomnoml" (default) or "plantuml" (experimental, brittle mostly for reasons out of our control), if the latter, arguments prefix, narrow, and code are ignored
browse	whether to debug step by step (block by block), can also be a vector of block ids, in this case browser() calls will be inserted at the start of these blocks
ref	the reference expression for flow_compare_runs()

Details

On some systems the output might sometimes display the box character when using the nomnoml engine, this is due to the system not recognizing the Braille character \u2800. This character is used to circumvent a shortcoming of the nomnoml library: lines can't start with a standard space and multiple subsequent spaces might be collapsed. To choose another character, set the option flow.indenter, for instance : options(flow.indenter = "\u00b7"). Setting the options(flow.svg = FALSE) might also help.

Value

depending on out :

- NULL (default) : flow_view() and flow_compare_runs() return a "flow_diagram" object, containing the diagram, the diagram's code and the data used to build the code. flow_run() returns the output of the call.
- An output path or a file extension : the path where the file is saved
- "data": a list of 2 data frames "nodes" and "edges"
- "code": A character vector of class "flow_code"

Examples

```
flow_view(rle)
flow_run(rle(c(1, 2, 2, 3)))
flow_compare_runs(rle(NULL), rle(c(1, 2, 2, 3)))
```

`flow_view_deps` *Show dependency graph of a function*

Description

[Experimental]

Usage

```
flow_view_deps(
  fun,
  max_depth = Inf,
  trim = NULL,
  promote = NULL,
  demote = NULL,
  hide = NULL,
  show_imports = c("functions", "packages", "none"),
  out = NULL,
  lines = TRUE,
  include_formals = TRUE
)
```

Arguments

`fun`	A function, can be of the form `fun`, `pkg::fun`, `pkg:::fun`, if in the form `fun`, the binding should be located in a package namespace or the global environment. It can also be a named list of functions, such as one you'd create with `dplyr::lst()`, for instance `lst(fun1, pkg::fun2)`.
`max_depth`	An integer, the maximum depth to display
`trim`	A vector or list of function names where the recursion will stop
`promote`	A vector or list of external functions to show as internal functions
`demote`	A vector or list of internal functions to show as external functions
`hide`	A vector or list of internal functions to completely remove from the chart
`show_imports`	Whether to show imported "functions", only "packages", or "none"
`out`	a path to save the diagram to. Special values "html", "htm", "png", "pdf", "jpg" and "jpeg" can be used to export the object to a temp file of the relevant format and open it, if a regular path is used the format will be guessed from the extension.
`lines`	Whether to show the number of lines of code next to the function name
`include_formals`	Whether to fetch dependencies in the default values of the function's arguments

Details

Exported objects are shown in blue, unexported objects are shown in yellow.

Regular expressions can be used in `trim`, `promote`, `demote` and `hide`, they will be used on function names in the form `pkg::fun` or `pkg:::fun` where `pkg` can be any package mentioned in these arguments, the namespace of the explored function, or any of the direct dependencies of the package. These arguments must be named, using the name "pattern". See examples below.

Value

`flow_view_deps()` returns a `"flow_diagram"` object by default, and the output path invisibly if `out` is not `NULL` (called for side effects).

Examples

```
flow_view_deps(here::i_am)
flow_view_deps(here::i_am, demote = "format_dr_here")
flow_view_deps(here::i_am, trim = "format_dr_here")
flow_view_deps(here::i_am, hide = "format_dr_here")
flow_view_deps(here::i_am, promote = "rprojroot::get_root_desc")
flow_view_deps(here::i_am, promote = c(pattern = ".*::g"))
flow_view_deps(here::i_am, promote = c(pattern = "rprojroot::.*"))
flow_view_deps(here::i_am, hide = c(pattern = "here:::s"))
```

---

flow_view_shiny *Visualize a shiny app's dependency graph*

Description

[Experimental] This function displays a shiny app's module structure, assuming it is built on top of module functions named a certain way (adjustable through the `pattern` argument) and calling each other. If you call for instance `flow_view_shiny()` on a function that runs the app and uses both the main server and ui functions, you'll display the full graph of server and ui modules.

Usage

```
flow_view_shiny(
  fun,
  max_depth = Inf,
  trim = NULL,
  promote = NULL,
  demote = NULL,
  hide = NULL,
  show_imports = c("functions", "packages", "none"),
  out = NULL,
  lines = TRUE,
  pattern = "(_ui)|(_server)|(Ui)|(Server)|(UI)|(SERVER)"
)
```

Arguments

fun	The function that runs the app
max_depth	An integer, the maximum depth to display
trim	A vector or list of function names where the recursion will stop
promote	A vector or list of external functions to show as internal functions
demote	A vector or list of internal functions to show as external functions
hide	A vector or list of internal functions to completely remove from the chart
show_imports	Whether to show imported "functions", only "packages", or "none"
out	a path to save the diagram to. Special values "html", "htm", "png", "pdf", "jpg" and "jpeg" can be used to export the object to a temp file of the relevant format and open it, if a regular path is used the format will be guessed from the extension.
lines	Whether to show the number of lines of code next to the function name
pattern	A regular expression used to detect ui and server functions

Details

It is wrapper around `flow_view_deps()` which demotes every object that is not a server function, a ui function or a function calling either. What is or isn't considered as a server or ui function depends on a regular expression provided through the `pattern` argument. For a more general way of displaying all dependencies (not focused on modules), use `flow_view_deps()`.

Value

A flow diagram object.

Examples

```
if (requireNamespace("esquisse", quietly = TRUE)) {
  flow_view_shiny(esquisse::esquisser, show_imports = "none")
}
```

`flow_view_source_calls`

Draw diagram of source dependencies

Description

Assuming a project where files source each other, draw their dependency graph.

Usage

```
flow_view_source_calls(
  paths = ".",
  recursive = TRUE,
  basename = TRUE,
  extension = FALSE,
  smart = TRUE,
  out = NULL
)
```

Arguments

paths	Paths to scripts or folders containing scripts By default explores the working directory.
recursive	Passed to `list.files()` when `paths` contains directories
basename	Whether to display only the base name of the script
extension	Whether to display the extension
smart	Whether to parse complex source calls for strings that look like script and match those to files found in `paths`
out	a path to save the diagram to. Special values "html", "htm", "png", "pdf", "jpg" and "jpeg" can be used to export the object to a temp file of the relevant format and open it, if a regular path is used the format will be guessed from the extension.

Details

This evaluates the `file` argument of `source` in the global environment, when this fails, as it might with constructs like `for (file in files) source(file)` the unevaluated argument is printed instead between backticks. Since this messes up the relationships in the graph, an warning is thus issued. In a case like `source(file.path(my_dir, "foo.R"))` defining `my_dir` will be enough to solve the issue. In the latter case, if `smart` is `TRUE`, the function will check in all the paths in scope if any script is named "foo.R" and will consider it if a single fitting candidate is found.
`c(c(1) c(c(1))`

Value

`flow_view_source_calls()` returns a `"flow_diagram"` object by default, and the output path invisibly if `out` is not `NULL` (called for side effects). `flow_run()` returns the output of the wrapped call.

flow_view_uses *Show graph of callers of a function*

Description

Experimental function that displays for a given object or function all functions that call it directly or indirectly.

Usage

```
flow_view_uses(x, pkg = NULL, out = NULL)
```

Arguments

x	An object
pkg	A package or environment to fetch callers from, by default fun's environment
out	a path to save the diagram to. Special values "html", "htm", "png", "pdf", "jpg" and "jpeg" can be used to export the object to a temp file of the relevant format and open it, if a regular path is used the format will be guessed from the extension.

Details

The function is not very robust yet, but already useful for many usecases.

Value

`flow_view_uses()` returns a `"flow_diagram"` object by default, and the output path invisibly if `out` is not `NULL` (called for side effects).

Examples

```
flow_view_uses(flow_run)
```

flow_view_vars *Draw the dependencies of variables in a function*

Description

[Experimental]

This draws the dependencies between variables. This function is useful to detect dead code and variable clusters. By default the variable is shown a new time when it's overwritten or modified, this can be changed by setting `expand` to `FALSE`.

Usage

```
flow_view_vars(
  x,
  expand = TRUE,
  refactor = c("refactored", "original"),
  out = NULL
)
```

Arguments

x	The function, script or expression to draw
expand	A boolean, if FALSE a variable name is only shown once, else (the default) it's repeated and suffixed with a number of *
refactor	If using 'refactor' package, whether to consider original or refactored code
out	a path to save the diagram to. Special values "html", "htm", "png", "pdf", "jpg" and "jpeg" can be used to export the object to a temp file of the relevant format and open it, if a regular path is used the format will be guessed from the extension.

Details

Colors and lines are to be understood as follows:

- The function is blue
- The arguments are green
- The variables starting as constants are yellow
- The dead code or pure side effect branches are orange and dashed
- dashed lines represent how variables are undirectly impacted by control flow conditions, for instance the expression `if(z == 1) x <- y` would give you a full arrow from `y` to `x` and a dashed arrow from `z` to `x`

`expand = TRUE` gives a sense of the chronology, and keep separate the unrelated uses of temp variables. `expand = FALSE` is more compact and shows you directly what variables might impact a given variable, and what variables it impacts.

This function will work best if the function doesn't draw from or assign to other environments and doesn't use `assign()` or `attach()`. The output might be polluted by variable names found in some lazily evaluated function arguments. We ignore variable names found in calls to `quote()` and `~` as well as nested function definitions, but complete robustness is probably impossible.

The diagram assumes that for / while / repeat loops were at least run once, if a value is modified in a branch of an if call (or both branches) and `expand` is `TRUE`, the modified variable(s) will point to a new one at the end of the `if` call.

Value

`flow_vars()` returns a `"flow_diagram"` object by default, and the output path invisibly if `out` is not `NULL` (called for side effects).

Examples

```
flow_view_vars(ave)
```

Index