Package 'flow'

March 8, 2022

Title View and Browse Code Using Flow Diagrams

Version 0.1.0

Description Visualize as flow diagrams the logic of functions, expressions or scripts in a static way or when running a call, and ease debugging. Advanced features include analogs to 'debug' and 'debugonce' to target specific functions to draw, an utility to draw the calls used in the tests of the package in a markdown report, and an utility to draw all the functions of one package in a markdown report.

License GPL-3

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

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

Encoding UTF-8

Suggests testthat (>= 2.1.0), covr, knitr, rmarkdown, esquisse, tidyselect

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

RoxygenNote 7.1.2

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Antoine Fabri [aut, cre]

Maintainer Antoine Fabri <antoine.fabri@gmail.com>

Repository CRAN

Date/Publication 2022-03-08 11:30:02 UTC

R topics documented:

`flow_debug` *Debug With Flow Diagrams*

Description

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

Usage

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

flow_debugonce(
  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() and debugonce(), flow_debug() and flow_debugonce() don'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
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 output (.html or .md), if left NULL a temp *html* file will be created and opened.
engine	either "nomnoml" (default) or "plantuml" (experimental), if the latter, arguments prefix, narrow, and code

Details

if pkg and out are both left NULL, a vignette diagrams.md will be built in the root, so that pkgdown::build_site will use it as an additional page. See also the vignette *"Build reports to document functions and unit tests"*.

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() or flow_debugonce(). 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_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 output (`.html` or `.md`), 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.

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
)
```

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), if the latter, arguments prefix, narrow, and code
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

On some system 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 nomnoml shortcoming: 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").

Value

flow_view() returns NULL invisibly, or the output path invisibly if out is not NULL (called for side effects). flow_run() returns the output of the wrapped call.

Examples

```
flow_view(rle)
flow_run(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
)
```

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
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

Details

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

Examples

```
flow_view_deps(flow_view_deps)
```

---

flow_view_shiny *Visualize a shiny app's dependency graph*

Description

[Experimental]

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

A 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.

Examples

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

---

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

Called for side effects

Examples

```
flow_view_vars(ave)
```

Index