Package: funviewR (via r-universe)

June 1, 2026

Title Visualize Function Call Dependencies in R Source Code

Version 0.1.1

Description Provides tools to analyze R source code and detect function definitions and their internal dependencies across multiple files. Creates interactive network visualizations using 'visNetwork' to display function call relationships, with detailed tooltips showing function arguments, return values, and documentation. Supports both individual files and directory-based analysis with automatic file detection. Useful for understanding code structure, identifying dependencies, and documenting R projects.

License GPL-3

URL https://github.com/deamonpog/funviewR

BugReports https://github.com/deamonpog/funviewR/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports codetools, visNetwork, igraph, htmltools, magrittr

Suggests htmlwidgets

Config/pak/sysreqs cmake libglpk-dev make libuv1-dev libxml2-dev

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

Date/Publication 2025-12-03 04:13:19 UTC

RemoteUrl https://github.com/deamonpog/funviewr

RemoteRef HEAD

RemoteSha 8041e4dbf6fa4c821f96bfd608241e57bfb24a58

Contents

`analyze_internal_dependencies_multi`
Analyze internal dependencies in R source files

Description

Parses multiple R files to identify function definitions and their internal dependencies. Creates a comprehensive map of which functions call which other functions within the analyzed codebase.

Usage

```
analyze_internal_dependencies_multi(file_paths)
```

Arguments

`file_paths`	Character vector of R file paths to analyze.

Details

This function uses `codetools::findGlobals()` to detect function calls within function bodies. Only functions defined within the analyzed files are tracked. External package functions and base R functions are not included in the dependency map.

Value

A list with the following components:

dependency_map Named list mapping function names to character vectors of their dependencies

env Environment containing all parsed functions

all_code_lines Character vector of all code lines from all files

function_file_map Named list mapping function names to their source file paths

duplicates List of functions defined in multiple files with their source locations

Examples

```
# Create temporary R files for demonstration
temp_file1 <- tempfile(fileext = ".R")
temp_file2 <- tempfile(fileext = ".R")

# Write sample R code to temporary files
writeLines(c(
  "add_numbers <- function(a, b) {",
```

```
  "  a + b",
  "}",
  "",
  "calculate_sum <- function(x) {",
  "  add_numbers(x, 10)",
  "}"
), temp_file1)

writeLines(c(
  "multiply <- function(a, b) {",
  "  a * b",
  "}",
  "",
  "process_data <- function(x) {",
  "  result <- add_numbers(x, 5)",
  "  multiply(result, 2)",
  "}"
), temp_file2)

# Analyze the files
dep_info <- analyze_internal_dependencies_multi(c(temp_file1, temp_file2))

# View the dependency map
print(dep_info$dependency_map)

# Check for duplicate function definitions
if (length(dep_info$duplicates) > 0) {
  message("Warning: Functions defined in multiple files:")
  print(dep_info$duplicates)
}

# Clean up
unlink(c(temp_file1, temp_file2))
```

`get_all_expressions` *Get all parsed expressions from an R file*

Description

Reads an R source file and parses it into a list of expressions. This is a utility function primarily used internally by `analyze_internal_dependencies_multi`.

Usage

```
get_all_expressions(file_path)
```

Arguments

`file_path`	Character string. Path to an R file to parse.

Value

A list of parsed expressions, or NULL if parsing fails.

Examples

```
# Create a temporary R file
temp_file <- tempfile(fileext = ".R")
writeLines(c(
  "my_function <- function(x) { x + 1 }",
  "another_function <- function(y) { y * 2 }"
), temp_file)

# Parse the file
exprs <- get_all_expressions(temp_file)
if (!is.null(exprs)) {
  print(length(exprs))
}

# Clean up
unlink(temp_file)
```

---

get_r_files *Get all R files from a directory*

Description

Recursively or non-recursively searches a directory for R source files matching a specified pattern. Returns full paths suitable for use with analyze_internal_dependencies_multi.

Usage

```
get_r_files(directory, recursive = FALSE, pattern = "\\.R$")
```

Arguments

directory	Character string. Path to directory containing R files.
recursive	Logical. If TRUE, search subdirectories recursively. Default is FALSE.
pattern	Character string. Regular expression pattern for filenames. Default is "\\.R$" (files ending in .R, case insensitive).

Value

Character vector of full paths to R files found in the directory. Returns an empty vector with a warning if no files are found.

Examples

```
# Create a temporary directory with R files
temp_dir <- tempfile()
dir.create(temp_dir)

# Create some R files
writeLines("f1 <- function(x) { x + 1 }", file.path(temp_dir, "file1.R"))
writeLines("f2 <- function(y) { y * 2 }", file.path(temp_dir, "file2.R"))

# Get all R files in the directory
files <- get_r_files(temp_dir)
print(files)

# Create subdirectory
subdir <- file.path(temp_dir, "subdir")
dir.create(subdir)
writeLines("f3 <- function(z) { z - 1 }", file.path(subdir, "file3.R"))

# Search recursively
files_recursive <- get_r_files(temp_dir, recursive = TRUE)
print(files_recursive)

# Clean up
unlink(temp_dir, recursive = TRUE)
```

---

plot_dependency_graph *Analyze and plot R code dependencies in one step*

Description

Convenience function that analyzes R files and directly returns the interactive dependency graph without exposing intermediate analysis results. Automatically detects whether paths are files or directories.

Usage

```
plot_dependency_graph(
  file_paths,
  include_disconnected = TRUE,
  recursive = FALSE
)
```

Arguments

`file_paths`	Character vector of R file paths, directory paths, or a mix. The function automatically detects files vs directories.
`include_disconnected`	Logical. If `FALSE`, exclude isolated nodes (functions with no dependencies) from the graph. Default is `TRUE`.

`recursive` Logical. If directories are encountered, search subdirectories recursively. Default is `FALSE`.

Details

This is a convenience wrapper around `analyze_internal_dependencies_multi` and `plot_interactive_dependency_gra`
It automatically:

- Detects whether each path is a file or directory
- Collects all R files from directories
- Analyzes function dependencies
- Creates an interactive visualization

Value

A `visNetwork` HTML widget displaying the interactive dependency graph. The graph can be saved using `htmlwidgets::saveWidget()`.

Examples

```
# Create temporary directory and files
temp_dir <- tempfile()
dir.create(temp_dir)

# Create sample R files
writeLines(c(
  "add <- function(a, b) { a + b }",
  "calc <- function(x) { add(x, 10) }"
), file.path(temp_dir, "math.R"))

writeLines(c(
  "process <- function(data) { add(data, 5) }"
), file.path(temp_dir, "process.R"))

# Analyze and plot - single file
graph <- plot_dependency_graph(file.path(temp_dir, "math.R"))

# Analyze directory
graph <- plot_dependency_graph(temp_dir)

# Exclude disconnected nodes
graph <- plot_dependency_graph(temp_dir, include_disconnected = FALSE)

# Clean up
unlink(temp_dir, recursive = TRUE)
```

`plot_interactive_dependency_graph` *Plot an interactive dependency graph*

Description

Creates an interactive network visualization of function dependencies using `visNetwork`. Nodes represent functions and edges represent function calls. The graph uses hierarchical layout with color-coding based on distance from the most-connected function.

Usage

```
plot_interactive_dependency_graph(dep_info, include_disconnected = TRUE)
```

Arguments

`dep_info`	List. Output from `analyze_internal_dependencies_multi` containing dependency information and function metadata.
`include_disconnected`	Logical. If `FALSE`, excludes nodes that are not connected to the center node (most-connected function). Default is `TRUE`.

Details

The visualization includes:

- **Node colors**: Functions are color-coded by their distance (number of hops) from the most-connected function
- **Node shapes**: Boxes for defined functions, ellipses for undefined/external functions
- **Tooltips**: Hover over nodes to see function arguments, return values, source file, and documentation
- **Interactive controls**: Zoom, pan, drag nodes, and navigation buttons

The center node (most-connected function) is fixed at position (0,0) and rendered in dark red. Other nodes are positioned using a force-directed layout.

Value

A `visNetwork` HTML widget displaying the interactive dependency graph. The widget can be displayed in RStudio Viewer, web browser, or saved to HTML using `htmlwidgets::saveWidget()`.

Examples

```
# Create temporary R files with sample code
temp_file <- tempfile(fileext = ".R")
writeLines(c(
  "helper_function <- function(x) { x * 2 }",
  "main_function <- function(a) { helper_function(a) + 1 }",
  "another_function <- function(b) { main_function(b) }"
), temp_file)

# Analyze the file
dep_info <- analyze_internal_dependencies_multi(temp_file)

# Create interactive graph
graph <- plot_interactive_dependency_graph(dep_info)

# Show only connected components
graph <- plot_interactive_dependency_graph(dep_info,
                                           include_disconnected = FALSE)

# Clean up
unlink(temp_file)
```

Index