Package 'putior'

June 19, 2025

Title ``Register In- and Outputs for Workflow Visualization''

Version 0.1.0

Author Philipp Thoss [aut, cre] (ORCID:
<https://orcid.org/0000-0002-4672-2792>)

Maintainer Philipp Thoss <ph.thoss@gmx.de>

Description Provides tools for extracting and processing structured annotations from 'R' and 'Python' source files to facilitate workflow visualization. The package scans source files for special 'PUT' annotations that define nodes, connections, and metadata within a data processing workflow. These annotations can then be used to generate visual representations of data flows and processing steps across polyglot software environments. Builds on concepts from literate programming Knuth (1984) <doi:10.1093/comjnl/27.2.97> and utilizes directed acyclic graph (DAG) theory for workflow representation Foraita, Spallek, and Zeeb (2014) <doi:10.1007/978-0-387-09834-0_65>. Diagram generation powered by 'Mermaid' Sveidqvist (2014) <https://mermaid.js.org/>.

Language en-US

License MIT + file LICENSE

URL https://pjt222.github.io/putior/, https://github.com/pjt222/putior

BugReports https://github.com/pjt222/putior/issues

Depends R (>= 3.5.0)

Imports tools

Suggests testthat (>= 3.0.0), knitr, rmarkdown, clipr, uuid, pkgdown

Encoding UTF-8

RoxygenNote 7.3.2

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2025-06-19 14:50:02 UTC

Contents

`get_diagram_themes` *Get available themes for put_diagram*

Description

Returns information about available color themes for workflow diagrams.

Usage

```
get_diagram_themes()
```

Value

Named list describing available themes

Examples

```
# See available themes
get_diagram_themes()

## Not run:
# Use a specific theme (requires actual workflow data)
workflow <- put("./src")
put_diagram(workflow, theme = "github")

## End(Not run)
```

`is_valid_put_annotation`
Validate PUT annotation syntax

Description

Test helper function to validate PUT annotation syntax

Usage

```
is_valid_put_annotation(line)
```

Arguments

line	Character string containing a PUT annotation

Value

Logical indicating if the annotation is valid

Examples

```
is_valid_put_annotation('#put name:"test", label:"Test"') # TRUE
is_valid_put_annotation("#put invalid syntax") # FALSE
```

---

put *Scan R and Python Files for PUT Annotations*

Description

Scans source files in a directory for PUT annotations that define workflow nodes, inputs, outputs, and metadata. Supports both R and Python files with flexible annotation syntax including single-line and multiline formats.

Usage

```
put(
  path,
  pattern = "\\.(R|r|py|sql|sh|jl)$",
  recursive = FALSE,
  include_line_numbers = FALSE,
  validate = TRUE
)
```

Arguments

path	Character string specifying the path to the folder containing files, or path to a single file
pattern	Character string specifying the file pattern to match. Default: "\.(R|r|py|sql|sh|jl)$" (R, Python, SQL, shell, Julia files)
recursive	Logical. Should subdirectories be searched recursively? Default: FALSE
include_line_numbers	Logical. Should line numbers be included in output? Default: FALSE
validate	Logical. Should annotations be validated for common issues? Default: TRUE

Value

A data frame containing file names and all properties found in annotations. Always includes columns: file_name, file_type, and any properties found in PUT annotations (typically: id, label, node_type, input, output). If include_line_numbers is TRUE, also includes line_number. Note: If output is not specified in an annotation, it defaults to the file name.

PUT Annotation Syntax

PUT annotations can be written in single-line or multiline format:

Single-line format: All parameters on one line

```
#put id:"node1", label:"Process Data", input:"data.csv", output:"result.csv"
```

Multiline format: Use backslash (\) for line continuation

```
#put id:"node1", label:"Process Data", \
#    input:"data.csv", \
#    output:"result.csv"
```

Benefits of multiline format:

- Compliance with code style guidelines (styler, lintr)
- Improved readability for complex workflows
- Easier maintenance of long file lists
- Better code organization and documentation

Syntax rules:

- End lines with backslash (\) to continue
- Each continuation line must start with # comment marker
- Properties are automatically joined with proper comma separation
- Works with all PUT formats: #put, # put, #putl, #put:

Examples

```
## Not run:
# Scan a directory for workflow annotations
workflow <- put("./src/")

# Scan recursively including subdirectories
workflow <- put("./project/", recursive = TRUE)

# Scan a single file
workflow <- put("./script.R")

# Include line numbers for debugging
workflow <- put("./src/", include_line_numbers = TRUE)

# Single-line PUT annotations (basic syntax):
# #put id:"load_data", label:"Load Dataset", node_type:"input", output:"data.csv"
# #put id:"process", label:"Clean Data", node_type:"process", input:"data.csv", output:"clean.csv"
#
# Multiline PUT annotations (for better code style compliance):
# Use backslash (\) at end of line to continue on next line
# #put id:"complex_process", label:"Complex Data Processing", \
```

```
# #    input:"file1.csv,file2.csv,file3.csv,file4.csv", \
# #    output:"results.csv"
#
# Multiline example with many files:
# #put id:"data_merger", \
# #    label:"Merge Multiple Data Sources", \
# #    node_type:"process", \
# #    input:"sales.csv,customers.csv,products.csv,inventory.csv", \
# #    output:"merged_dataset.csv"
#
# All PUT formats support multiline syntax:
# # put id:"style1", label:"Standard" \     # Space after #
# #put| id:"style2", label:"Pipe" \        # Pipe separator
# #put: id:"style3", label:"Colon" \       # Colon separator

## End(Not run)
```

`put_diagram` *Create Mermaid Diagram from PUT Workflow*

Description

Generates a Mermaid flowchart diagram from putior workflow data, showing the flow of data through your analysis pipeline.

Usage

```
put_diagram(
  workflow,
  output = "console",
  file = "workflow_diagram.md",
  title = NULL,
  direction = "TD",
  node_labels = "label",
  show_files = FALSE,
  show_artifacts = FALSE,
  show_workflow_boundaries = TRUE,
  style_nodes = TRUE,
  theme = "light"
)
```

Arguments

`workflow`	Data frame returned by `put()` containing workflow nodes
`output`	Character string specifying output format. Options: • "console" - Print to console (default) • "file" - Save to file specified by `file` parameter • "clipboard" - Copy to clipboard (if available)

- "raw" - Return raw mermaid code without markdown fences (for knitr/pkgdown)

file	Character string specifying output file path (used when output = "file")
title	Character string for diagram title (optional)
direction	Character string specifying diagram direction. Options: "TD" (top-down), "LR" (left-right), "BT" (bottom-top), "RL" (right-left)
node_labels	Character string specifying what to show in nodes: "name" (node IDs), "label" (descriptions), "both" (ID: label)
show_files	Logical indicating whether to show file connections
show_artifacts	Logical indicating whether to show data files as nodes. When TRUE, creates nodes for all input/output files, not just script connections. This provides a complete view of the data flow including terminal outputs.
show_workflow_boundaries	Logical indicating whether to apply special styling to nodes with node_type "start" and "end". When TRUE, these nodes get distinctive workflow boundary styling (icons, colors). When FALSE, they render as regular nodes.
style_nodes	Logical indicating whether to apply styling based on node_type
theme	Character string specifying color theme. Options: "light" (default), "dark", "auto" (GitHub adaptive), "minimal", "github"

Value

Character string containing the mermaid diagram code

Examples

```
## Not run:
# Basic usage - shows only script connections
workflow <- put("./src/")
put_diagram(workflow)

# Show all data artifacts as nodes (complete data flow)
put_diagram(workflow, show_artifacts = TRUE)

# Show artifacts with file labels on connections
put_diagram(workflow, show_artifacts = TRUE, show_files = TRUE)

# Show workflow boundaries with special start/end styling
put_diagram(workflow, show_workflow_boundaries = TRUE)

# Disable workflow boundaries (start/end nodes render as regular)
put_diagram(workflow, show_workflow_boundaries = FALSE)

# GitHub-optimized theme for README files
put_diagram(workflow, theme = "github")

# Save to file with artifacts enabled
put_diagram(workflow, show_artifacts = TRUE, output = "file", file = "workflow.md")
```

```
# For use in knitr/pkgdown - returns raw mermaid code
# Use within a code chunk with results='asis'
cat("```mermaid\n", put_diagram(workflow, output = "raw"), "\n```\n")

## End(Not run)
```

`split_file_list` *Split comma-separated file list*

Description

Split comma-separated file list

Usage

```
split_file_list(file_string)
```

Arguments

`file_string`	Comma-separated file names

Value

Character vector of individual file names

Index