Package: putior (via r-universe)

May 18, 2026

Title Generate Workflow Diagrams from Code Annotations

Version 0.2.0

Description Provides tools for extracting and processing structured annotations from source files in 30+ programming languages to facilitate workflow visualization. The package scans source files for special 'PUT' annotations that define nodes, connections, and metadata within a data processing workflow. Supports R, Python, SQL, JavaScript, TypeScript, Go, Rust, Java, and more with automatic comment syntax detection. 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 (>= 4.1.0)

Imports tools

Suggests testthat (>= 3.0.0), knitr, rmarkdown, clipr, uuid, pkgdown, logger, shiny, shinyAce, mcptools, ellmer, plumber2

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Config/testthat/edition 3

VignetteBuilder knitr

Repository https://pjt222.r-universe.dev

Date/Publication 2026-03-19 11:44:49 UTC

RemoteUrl https://github.com/pjt222/putior

RemoteRef HEAD

RemoteSha 9cec4b6030630205f75c5e7b6024fb8345b91b8b

Contents

ext_to_language . . . 2
get_comment_prefix . . . 3
get_detection_patterns . . . 4
get_diagram_themes . . . 5
get_supported_extensions . . . 6
is_valid_put_annotation . . . 6
list_supported_languages . . . 7
print.putior_theme . . . 8
print.putior_workflow . . . 8
put . . . 9
put_auto . . . 11
put_diagram . . . 13
put_generate . . . 16
put_merge . . . 18
put_theme . . . 19
putior_acp_manifest . . . 21
putior_acp_server . . . 21
putior_guide . . . 23
putior_help . . . 24
putior_mcp_server . . . 25
putior_mcp_tools . . . 27
run_sandbox . . . 28
set_putior_log_level . . . 29
split_file_list . . . 30
summary.putior_workflow . . . 31

Index **32**

`ext_to_language` *Get Language Name from File Extension*

Description

Converts a file extension to a standardized language name used internally for detection pattern lookup.

Usage

```
ext_to_language(ext)
```

Arguments

ext	Character string of the file extension (without dot)

Value

Character string of the language name, or NULL if not supported

See Also

Other language-support: get_comment_prefix(), get_detection_patterns(), get_supported_extensions(), list_supported_languages()

Examples

```
ext_to_language("r")     # Returns "r"
ext_to_language("py")    # Returns "python"
ext_to_language("sql")   # Returns "sql"
ext_to_language("js")    # Returns "javascript"
ext_to_language("xyz")   # Returns NULL
```

---

get_comment_prefix *Get Comment Prefix for File Extension*

Description

Returns the single-line comment prefix for a given file extension. Falls back to # for unknown extensions.

Usage

```
get_comment_prefix(ext)
```

Arguments

ext	Character string of the file extension (without dot)

Value

Character string of the comment prefix

See Also

Other language-support: ext_to_language(), get_detection_patterns(), get_supported_extensions(), list_supported_languages()

Examples

```
get_comment_prefix("r")    # Returns "#"
get_comment_prefix("sql")  # Returns "--"
get_comment_prefix("js")   # Returns "//"
get_comment_prefix("tex")  # Returns "%"
get_comment_prefix("xyz")  # Returns "#" (fallback)
```

`get_detection_patterns` *Get Detection Patterns for a Language*

Description

Returns the detection patterns for identifying inputs, outputs, and dependencies in source code files for a specific programming language.

Usage

```
get_detection_patterns(language = "r", type = NULL)
```

Arguments

`language`	Character string specifying the language. Options: "r", "python", "sql", "shell", "julia", "javascript", "typescript", "go", "rust", "java", "c", "cpp", "matlab", "ruby", "lua", "wgsl"
`type`	Optional character string to filter by detection type. Options: "input", "output", "dependency". If NULL (default), returns all.

Value

A list of patterns with the following structure:

- input: List of patterns for detecting file inputs
- output: List of patterns for detecting file outputs
- dependency: List of patterns for detecting script dependencies

Each pattern contains:

- regex: Regular expression to match the function call
- arg_position: Position of the file path argument (1-indexed)
- arg_name: Named argument for file path (alternative to position)
- description: Human-readable description

See Also

Other language-support: `ext_to_language()`, `get_comment_prefix()`, `get_supported_extensions()`, `list_supported_languages()`

Examples

```
# Get all R patterns
patterns <- get_detection_patterns("r")

# Get only input patterns for Python
input_patterns <- get_detection_patterns("python", type = "input")

# Get dependency patterns for R
dep_patterns <- get_detection_patterns("r", type = "dependency")
```

`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

See Also

Other utilities: `is_valid_put_annotation()`, `split_file_list()`

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)
```

get_supported_extensions *Get All Supported File Extensions*

Description

Returns a character vector of all file extensions supported by putior's annotation parsing system.

Usage

```
get_supported_extensions()
```

Value

Character vector of supported file extensions (without dots)

See Also

Other language-support: `ext_to_language()`, `get_comment_prefix()`, `get_detection_patterns()`, `list_supported_languages()`

Examples

```
get_supported_extensions()
```

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

See Also

Other utilities: `get_diagram_themes()`, `split_file_list()`

Examples

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

`list_supported_languages`

List All Supported Languages

Description

Returns a character vector of all programming languages that can have PUT annotations parsed by putior. Note that detection patterns (for `put_auto`) may only be available for a subset of these languages.

Usage

```
list_supported_languages(detection_only = FALSE)
```

Arguments

`detection_only`	Logical. If TRUE, only return languages with detection pattern support. Default: FALSE

Value

Character vector of supported language names

See Also

Other language-support: `ext_to_language()`, `get_comment_prefix()`, `get_detection_patterns()`, `get_supported_extensions()`

Examples

```
# All languages with annotation parsing support
list_supported_languages()

# Only languages with auto-detection patterns
list_supported_languages(detection_only = TRUE)
```

print.putior_theme *Print a putior_theme Object*

Description

Print a putior_theme Object

Usage

```
## S3 method for class 'putior_theme'
print(x, ...)
```

Arguments

x	A putior_theme object.
...	Additional arguments (ignored).

Value

The object, invisibly.

See Also

Other core-workflow: print.putior_workflow(), put(), put_diagram(), put_theme(), summary.putior_workflow()

print.putior_workflow *Print a putior workflow*

Description

Displays a concise summary of a putior workflow data frame.

Usage

```
## S3 method for class 'putior_workflow'
print(x, ...)
```

Arguments

x	A putior_workflow object
...	Additional arguments (ignored)

Value

Invisibly returns x

See Also

Other core-workflow: print.putior_theme(), put(), put_diagram(), put_theme(), summary.putior_workflow()

---

put *Scan Source Files for PUT Annotations*

Description

Scans source files in a directory for PUT annotations that define workflow nodes, inputs, outputs, and metadata. Supports multiple programming languages with their native comment syntax, including single-line and multiline formats.

Usage

```
put(
  path,
  pattern = NULL,
  recursive = TRUE,
  include_line_numbers = FALSE,
  validate = TRUE,
  exclude = NULL,
  log_level = NULL
)
```

Arguments

path	Character string specifying the path to the folder containing files, or path to a single file
pattern	Regex pattern for filtering files (e.g., \\.R$ for R files only).
recursive	Logical. Should subdirectories be searched recursively? Default: TRUE
include_line_numbers	Logical. Should line numbers be included in output? Default: FALSE
validate	Logical. Should annotations be validated for common issues? Default: TRUE
exclude	Character vector of regex patterns. Files whose full path matches any pattern are excluded from scanning. Default: NULL (no exclusion). Example: exclude = c("_meta\.R$", "deprecated/")
log_level	Character string specifying log verbosity for this call. Overrides the global option putior.log_level when specified. Options: "DEBUG", "INFO", "WARN", "ERROR". See set_putior_log_level.

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). Valid `node_type` values: `"input"`, `"process"` (default), `"output"`, `"decision"`, `"start"`, `"end"`. An additional type `"artifact"` is used internally by `put_diagram()` for data file nodes and should not be set manually. 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.

Supported Languages

PUT annotations work with any language by using the appropriate comment prefix:

- **Hash (#)**: R, Python, Shell, Julia, Ruby, Perl, YAML
- **Dash (–)**: SQL, Lua, Haskell
- **Slash (//)**: JavaScript, TypeScript, C/C++, Java, Go, Rust, Swift, Kotlin, C#
- **Percent (%)**: MATLAB, LaTeX

PUT Annotation Syntax

PUT annotations can be written in single-line or multiline format. The comment prefix is determined automatically by file extension.

Single-line format (various languages):

```
# put id:"node1", label:"Process"       # R/Python
--put id:"node1", label:"Query"        -- SQL
//put id:"node1", label:"Handler"      // JavaScript
```

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 the appropriate comment marker
- Properties are automatically joined with proper comma separation
- Works with all PUT formats: prefix+put, prefix + put, prefix+putl, prefix+put:

See Also

Other core-workflow: print.putior_theme(), print.putior_workflow(), put_diagram(), put_theme(), summary.putior_workflow()

Examples

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

# Scan top-level only (opt out of recursion)
workflow <- put("./project/", recursive = FALSE)

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

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

# Scan JavaScript/TypeScript files
workflow <- put("./frontend/", pattern = "\\.(js|ts|jsx|tsx)$")

# Scan SQL files
workflow <- put("./sql/", pattern = "\\.sql$")

# Single-line PUT annotations (various languages):
# R/Python: # put id:"load_data", label:"Load Dataset"
# SQL:      --put id:"query", label:"Execute Query"
# JS/TS:    //put id:"handler", label:"API Handler"
# MATLAB:   %put id:"compute", label:"Compute Results"
#
# Multiline PUT annotations work the same across languages:
# # put id:"complex_process", label:"Complex Processing", \
# #    input:"file1.csv,file2.csv", \
# #    output:"results.csv"
#
# --put id:"etl_job", label:"ETL Process", \
# --    input:"source_table", \
# --    output:"target_table"

## End(Not run)
```

---

put_auto *Auto-Annotation Functions for putior*

Description

Functions for automatically detecting workflow elements from code analysis, generating PUT annotation comments, and merging manual with auto-detected annotations.

Analyzes source code files to automatically detect workflow elements including inputs, outputs, and dependencies without requiring explicit PUT annotations. This is similar to how roxygen2 auto-generates documentation skeletons.

Usage

```
put_auto(
  path,
  pattern = NULL,
  recursive = TRUE,
  detect_inputs = TRUE,
  detect_outputs = TRUE,
  detect_dependencies = TRUE,
  include_line_numbers = FALSE,
  exclude = NULL,
  log_level = NULL
)
```

Arguments

path	Character string specifying the path to the folder containing files, or path to a single file
pattern	Regex pattern for filtering files (e.g., \\.R$ for R files only).
recursive	Logical. Should subdirectories be searched recursively? Default: TRUE
detect_inputs	Logical. Should file inputs be detected? Default: TRUE
detect_outputs	Logical. Should file outputs be detected? Default: TRUE
detect_dependencies	Logical. Should script dependencies (source calls) be detected? Default: TRUE
include_line_numbers	Logical. Should line numbers be included? Default: FALSE
exclude	Character vector of regex patterns. Files whose full path matches any pattern are excluded from scanning. Default: NULL (no exclusion).
log_level	Character string specifying log verbosity for this call. Overrides the global option putior.log_level when specified. Options: "DEBUG", "INFO", "WARN", "ERROR". See set_putior_log_level.

Value

A data frame in the same format as put(), containing:

- file_name: Name of the source file
- file_path: Full path to the file
- file_type: File extension (r, py, sql, etc.)
- id: Auto-generated node identifier (based on file name)
- label: Human-readable label (file name without extension)
- input: Comma-separated list of detected input files

- output: Comma-separated list of detected output files
- node_type: Inferred node type (input/process/output)

This format is directly compatible with put_diagram().

See Also

put for manual annotation extraction, put_generate for generating annotation comments, put_merge for combining manual and auto-detected annotations

Other auto-annotation: put_generate(), put_merge()

Examples

```
## Not run:
# Auto-detect workflow from a directory
workflow <- put_auto("./src/")
put_diagram(workflow)

# Auto-detect with line numbers
workflow <- put_auto("./scripts/", include_line_numbers = TRUE)

# Only detect outputs (useful for finding data products)
outputs_only <- put_auto("./analysis/", detect_inputs = FALSE)

## 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",
  palette = NULL,
  show_source_info = FALSE,
  source_info_style = "inline",
  enable_clicks = FALSE,
  click_protocol = "vscode",
  log_level = NULL
)
```

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 light theme with bright colors (default) • "dark" - Dark theme for dark mode environments • "auto" - GitHub-adaptive with solid colors for both modes • "minimal" - Grayscale professional, print-friendly • "github" - Optimized for GitHub README files • "viridis" - Colorblind-safe (purple-blue-green-yellow) • "magma" - Colorblind-safe warm (purple-red-yellow) • "plasma" - Colorblind-safe vibrant (purple-pink-yellow) • "cividis" - Colorblind-safe for deuteranopia/protanopia (blue-yellow) The viridis family themes are perceptually uniform and tested for accessibility.

palette — Optional putior_theme object created by put_theme() for custom node colors. When provided, this overrides the theme parameter. Default is NULL (use theme).

show_source_info — Logical indicating whether to display source file information in diagram nodes. When TRUE, each node shows its originating file name. Default is FALSE for backward compatibility.

source_info_style — Character string specifying how to display source info:

- "inline" - Append file name to node labels (default)
- "subgraph" - Group nodes by source file into Mermaid subgraphs

enable_clicks — Logical indicating whether to add click directives to nodes. When TRUE, nodes become clickable links that open the source file in an editor. Default is FALSE for backward compatibility.

click_protocol — Character string specifying the URL protocol for clickable nodes:

- "vscode" - VS Code editor (vscode://file/path:line) (default)
- "file" - Standard file:// protocol
- "rstudio" - RStudio IDE (rstudio://open-file?path=)

log_level — Character string specifying log verbosity for this call. Overrides the global option putior.log_level when specified. Options: "DEBUG", "INFO", "WARN", "ERROR". See set_putior_log_level.

Value

Character string containing the mermaid diagram code

See Also

Other core-workflow: print.putior_theme(), print.putior_workflow(), put(), put_theme(), summary.putior_workflow()

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")

# Show source file info inline in nodes
put_diagram(workflow, show_source_info = TRUE)

# Group nodes by source file using subgraphs
put_diagram(workflow, show_source_info = TRUE, source_info_style = "subgraph")

# Enable clickable nodes (opens in VS Code)
put_diagram(workflow, enable_clicks = TRUE)

# Enable clickable nodes with RStudio protocol
put_diagram(workflow, enable_clicks = TRUE, click_protocol = "rstudio")

# Use a custom color palette
my_theme <- put_theme(base = "dark",
  input = c(fill = "#1a5276", stroke = "#2e86c1", color = "#ffffff"))
put_diagram(workflow, palette = my_theme)

## End(Not run)
```

`put_generate` *Generate PUT Annotation Comments*

Description

Analyzes source code files and generates suggested PUT annotation comments based on detected inputs, outputs, and dependencies. This is similar to how roxygen2 generates documentation skeletons.

Usage

```
put_generate(
  path,
  pattern = NULL,
  recursive = TRUE,
  output = "console",
  insert = FALSE,
  style = "multiline",
```

```
  exclude = NULL,
  log_level = NULL
)
```

Arguments

path	Character string specifying the path to a file or directory
pattern	Regex pattern for filtering files (e.g., \\.R$ for R files only).
recursive	Logical. Should subdirectories be searched recursively? Default: TRUE
output	Character string specifying output destination: • "console" - Print to console (default) • "raw" - Return as string without printing • "clipboard" - Copy to clipboard (requires clipr package) • "file" - Write to files with .put suffix
insert	Logical. If TRUE, insert annotations directly into source files at the top. Use with caution. Default: FALSE
style	Character string specifying annotation style: • "single" - Single-line annotations • "multiline" - Multiline annotations with backslash continuation Default: "multiline"
exclude	Character vector of regex patterns. Files whose full path matches any pattern are excluded. Default: NULL (no exclusion).
log_level	Character string specifying log verbosity for this call. Overrides the global option putior.log_level when specified. Options: "DEBUG", "INFO", "WARN", "ERROR". See set_putior_log_level.

Value

Invisibly returns a character vector of generated annotations. Side effects depend on the output parameter.

See Also

put_auto for direct workflow detection, put for extracting existing annotations

Other auto-annotation: put_auto(), put_merge()

Examples

```
## Not run:
# Print suggested annotations to console
put_generate("./analysis.R")

# Copy to clipboard for easy pasting
put_generate("./scripts/", output = "clipboard")

# Generate multiline style annotations
```

```
put_generate("./src/", style = "multiline")

# Insert annotations directly into files (use with caution)
put_generate("./new_script.R", insert = TRUE)

## End(Not run)
```

`put_merge` *Merge Manual and Auto-Detected Annotations*

Description

Combines manually written PUT annotations with auto-detected workflow elements, allowing flexible strategies for handling conflicts and supplementing information.

Usage

```
put_merge(
  path,
  pattern = NULL,
  recursive = TRUE,
  merge_strategy = "manual_priority",
  include_line_numbers = FALSE,
  exclude = NULL,
  log_level = NULL
)
```

Arguments

`path`	Character string specifying the path to a file or directory
`pattern`	Regex pattern for filtering files (e.g., `\\.R$` for R files only).
`recursive`	Logical. Should subdirectories be searched recursively? Default: TRUE
`merge_strategy`	Character string specifying how to merge: • "manual_priority" - Manual annotations override auto-detected (default) • "supplement" - Auto fills in missing input/output fields only • "union" - Combine all detected I/O from both sources
`include_line_numbers`	Logical. Should line numbers be included? Default: FALSE
`exclude`	Character vector of regex patterns. Files whose full path matches any pattern are excluded. Default: NULL (no exclusion).
`log_level`	Character string specifying log verbosity for this call. Overrides the global option `putior.log_level` when specified. Options: "DEBUG", "INFO", "WARN", "ERROR". See `set_putior_log_level`.

Value

A data frame in the same format as put(), containing merged workflow information from both manual and auto-detected sources.

See Also

put for manual annotation extraction, put_auto for auto-detection

Other auto-annotation: put_auto(), put_generate()

Examples

```
## Not run:
# Merge with manual annotations taking priority
workflow <- put_merge("./src/")
put_diagram(workflow)

# Supplement manual annotations with auto-detected I/O
workflow <- put_merge("./scripts/", merge_strategy = "supplement")

# Combine all inputs/outputs from both sources
workflow <- put_merge("./analysis/", merge_strategy = "union")

## End(Not run)
```

---

put_theme *Create a Custom Theme Palette for Workflow Diagrams*

Description

Constructs a custom color palette for use with put_diagram(). Specify colors for any subset of node types; unspecified types inherit from the base theme.

Usage

```
put_theme(
  base = "light",
  input = NULL,
  process = NULL,
  output = NULL,
  decision = NULL,
  artifact = NULL,
  start = NULL,
  end = NULL
)
```

Arguments

base	Character string naming the base theme to inherit from. Must be one of the valid themes returned by get_diagram_themes(). Default: "light".
input	Named character vector c(fill, stroke, color) for input nodes.
process	Named character vector c(fill, stroke, color) for process nodes.
output	Named character vector c(fill, stroke, color) for output nodes.
decision	Named character vector c(fill, stroke, color) for decision nodes.
artifact	Named character vector c(fill, stroke, color) for artifact nodes.
start	Named character vector c(fill, stroke, color) for start nodes.
end	Named character vector c(fill, stroke, color) for end nodes.

Details

Each node type accepts a named character vector with keys fill, stroke, and color (text color). All values must be valid hex colors (e.g., "#1a5276").

Value

An object of class putior_theme (a named list of CSS style strings, one per node type), suitable for passing to put_diagram(palette = ...).

See Also

Other core-workflow: print.putior_theme(), print.putior_workflow(), put(), put_diagram(), summary.putior_workflow()

Examples

```
# Override only input node colors, inherit rest from dark theme
my_theme <- put_theme(base = "dark",
  input = c(fill = "#1a5276", stroke = "#2e86c1", color = "#ffffff"))

# Full custom palette
custom <- put_theme(
  input   = c(fill = "#264653", stroke = "#2a9d8f", color = "#ffffff"),
  process = c(fill = "#e9c46a", stroke = "#f4a261", color = "#000000"),
  output  = c(fill = "#e76f51", stroke = "#264653", color = "#ffffff"))

## Not run:
workflow <- put("./src/")
put_diagram(workflow, palette = my_theme)

## End(Not run)
```

putior_acp_manifest *Get putior ACP Agent Manifest*

Description

Returns the agent definition for ACP discovery. This manifest describes putior's capabilities to other AI agents.

Usage

```
putior_acp_manifest()
```

Value

A list containing the agent manifest with name, description, and metadata about capabilities.

See Also

Other integration: `putior_acp_server()`, `putior_guide()`, `putior_help()`, `putior_mcp_server()`, `putior_mcp_tools()`

Examples

```
# Get the agent manifest
manifest <- putior_acp_manifest()
print(manifest$name)
print(manifest$description)
```

putior_acp_server *Start putior ACP Server*

Description

Starts an ACP (Agent Communication Protocol) REST server that exposes putior as an agent that other AI agents can discover and call.

Usage

```
putior_acp_server(host = "127.0.0.1", port = 8080L)
```

Arguments

host	Character string specifying the host address. Default: "127.0.0.1" (localhost only)
port	Integer specifying the port number. Default: 8080

Details

The ACP server exposes the following endpoints:

GET /agents Returns the agent manifest for discovery

POST /runs Execute a putior operation

GET /runs/:run_id Get the status/result of a previous run

Value

This function does not return; it runs the server until terminated.

Message Format

POST /runs expects a JSON body with this structure:

```
{
  "input": [
    {
      "role": "user",
      "parts": [
        {
          "content": "scan ./R/ for workflow annotations",
          "content_type": "text/plain"
        }
      ]
    }
  ],
  "session_id": "optional-session-id"
}
```

Supported Operations

The agent understands natural language requests for:

- **scan**: "Scan ./R/ for PUT annotations"
- **diagram**: "Generate a diagram for ./R/"
- **auto**: "Auto-detect workflow from ./src/"
- **generate**: "Generate annotation suggestions for ./R/"
- **merge**: "Merge manual and auto annotations in ./R/"
- **help**: "Help with annotation syntax"
- **guide**: "What are your capabilities?"

Testing

Test the server with curl:

```
# Discover agents
curl http://localhost:8080/agents

# Execute a scan
curl -X POST http://localhost:8080/runs \
  -H "Content-Type: application/json" \
  -d '{"input": [{"role": "user", "parts": [{"content": "scan ./R/"}]}]}'
```

See Also

putior_acp_manifest for the agent definition, putior_mcp_server for MCP (Model-to-tool) integration

Other integration: putior_acp_manifest(), putior_guide(), putior_help(), putior_mcp_server(), putior_mcp_tools()

Examples

```
## Not run:
# Start ACP server on default port
putior_acp_server()

# Start on custom host/port
putior_acp_server(host = "0.0.0.0", port = 9000)

## End(Not run)
```

---

putior_guide *Access putior Guide for AI Assistants*

Description

Provides structured reference documentation for AI coding assistants (Claude Code, GitHub Copilot, etc.) to help users with putior.

Usage

```
putior_guide(topic = NULL, output = c("console", "raw", "clipboard"))
```

Arguments

topic	Character string specifying section. NULL for full content. Options: "quickstart", "syntax", "languages", "functions", "patterns", "examples", or NULL (all)
output	Output format: "console" (default), "raw", or "clipboard"

Details

For step-by-step procedures, see the agent-almanac repository, which provides 6 skills for the complete putior workflow.

Value

Invisibly returns content as character vector. With output="raw", returns as single string.

See Also

Other integration: `putior_acp_manifest()`, `putior_acp_server()`, `putior_help()`, `putior_mcp_server()`, `putior_mcp_tools()`

Examples

```
# Show full guide
putior_guide()

# Show specific topic
putior_guide("quick-start")

# Get raw markdown for AI consumption
guide_md <- putior_guide(output = "raw")
```

`putior_help` *Quick Reference Help for putior*

Description

Provides quick-reference help for common putior tasks and syntax. Call without arguments to see available topics.

Usage

```
putior_help(topic = NULL)
```

Arguments

`topic`	Character string specifying the help topic. Options: • NULL (default) - Show available topics • "annotation" or "annotations" - Show annotation syntax reference • "themes" - Show available diagram themes • "languages" - Show supported languages and comment prefixes • "node_types" - Show available node types • "patterns" - Show how to use detection patterns • "examples" - Show quick examples • "guide" - Show AI assistant guide reference

Value

Invisibly returns NULL. Prints help content to the console.

See Also

Other integration: putior_acp_manifest(), putior_acp_server(), putior_guide(), putior_mcp_server(), putior_mcp_tools()

Examples

```
# Show available topics
putior_help()

# Show annotation syntax
putior_help("annotation")

# Show supported languages
putior_help("languages")

# Show available themes
putior_help("themes")

# Show node types
putior_help("node_types")

# Show detection patterns info
putior_help("patterns")

# Show quick examples
putior_help("examples")

# Show AI assistant guide reference
putior_help("guide")
```

---

putior_mcp_server *Start putior MCP Server*

Description

Starts an MCP server that exposes putior functions as tools for AI assistants. This enables AI coding assistants (Claude Code, Claude Desktop) to directly call workflow annotation and diagram generation functions.

Usage

```
putior_mcp_server(type = c("stdio", "http"), host = "127.0.0.1", port = 8080L)
```

Arguments

type	Character string specifying the transport type: • "stdio" (default) - Standard input/output transport • "http" - HTTP transport
host	Character string specifying the host for HTTP transport. Default: "127.0.0.1"
port	Integer specifying the port for HTTP transport. Default: 8080

Details

The MCP server exposes the following putior functions as tools:

- put - Scan files for PUT annotations
- put_diagram - Generate Mermaid diagrams
- put_auto - Auto-detect workflow from code
- put_generate - Generate annotation suggestions
- put_merge - Merge manual + auto annotations
- get_comment_prefix - Get comment prefix for extension
- get_supported_extensions - List supported extensions
- list_supported_languages - List supported languages
- get_detection_patterns - Get auto-detection patterns
- get_diagram_themes - List available themes
- putior_guide - AI assistant documentation
- putior_help - Quick reference help
- set_putior_log_level - Configure logging
- is_valid_put_annotation - Validate annotation syntax
- split_file_list - Parse file lists
- ext_to_language - Extension to language name

Value

This function does not return; it runs the MCP server until terminated.

Configuration

Claude Code (WSL/Linux/macOS):

```
claude mcp add putior -- Rscript -e "putior::putior_mcp_server()"
```

Claude Desktop (Windows): Add to %APPDATA%\Claude\claude_desktop_config.json:

```
{
  "mcpServers": {
    "putior": {
      "command": "Rscript",
      "args": ["-e", "putior::putior_mcp_server()"]
    }
  }
}
```

See Also

putior_mcp_tools for accessing the tool definitions

Other integration: putior_acp_manifest(), putior_acp_server(), putior_guide(), putior_help(), putior_mcp_tools()

Examples

```
## Not run:
# Start MCP server with default stdio transport
putior_mcp_server()

# Start MCP server with HTTP transport
putior_mcp_server(type = "http", port = 8080)

## End(Not run)
```

---

putior_mcp_tools *Get putior MCP Tool Definitions*

Description

Returns a list of ellmer tool definitions for all putior functions that can be exposed via MCP. This function is primarily used internally by putior_mcp_server, but can also be used directly for inspection or custom MCP server implementations.

Usage

```
putior_mcp_tools(include = NULL, exclude = c("run_sandbox"))
```

Arguments

include	Character vector of tool names to include. If NULL (default), all available tools are included.
exclude	Character vector of tool names to exclude. Default excludes "run_sandbox" since Shiny apps cannot run via MCP.

Value

A list of ellmer tool definitions suitable for use with `mcptools::mcp_server()`.

See Also

`putior_mcp_server` for starting the MCP server

Other integration: `putior_acp_manifest()`, `putior_acp_server()`, `putior_guide()`, `putior_help()`, `putior_mcp_server()`

Examples

```
## Not run:
# Get all putior MCP tools
tools <- putior_mcp_tools()

# Get specific tools only
tools <- putior_mcp_tools(include = c("put", "put_diagram"))

# Exclude specific tools
tools <- putior_mcp_tools(exclude = c("run_sandbox", "putior_help"))

## End(Not run)
```

---

run_sandbox *Launch putior Interactive Sandbox*

Description

Opens an interactive Shiny application for experimenting with PUT annotations and workflow diagrams. Users can paste or type annotated code, adjust diagram settings, and see real-time diagram generation without installing the package locally.

Usage

```
run_sandbox()
```

Details

The sandbox app allows you to:

- Enter annotated code with PUT comments
- Simulate multiple files using file markers
- Customize diagram appearance (theme, direction, etc.)
- View extracted workflow data
- Copy or download generated Mermaid code

Value

Launches the Shiny app in the default browser. Returns invisibly.

See Also

put, put_diagram

Other interactive: set_putior_log_level()

Examples

```
## Not run:
# Launch the interactive sandbox
run_sandbox()

## End(Not run)
```

---

set_putior_log_level *Set putior Log Level*

Description

Configure the logging verbosity for putior functions. Higher verbosity levels provide more detailed information about internal operations, which is useful for debugging annotation parsing, workflow detection, and diagram generation.

Usage

```
set_putior_log_level(level = "WARN")
```

Arguments

level	Character string specifying the log level: • "DEBUG" - Fine-grained internal operations (file-by-file, pattern matching) • "INFO" - Progress milestones (scan started, nodes found, diagram generated) • "WARN" - Issues that don't stop execution (validation issues, missing deps) - default • "ERROR" - Fatal issues (via existing stop() calls)

Value

Invisibly returns the previous log level

See Also

Other interactive: run_sandbox()

Examples

```
# Save current level, change, then restore
old_level <- set_putior_log_level("DEBUG")
getOption("putior.log_level")
set_putior_log_level(old_level)
```

`split_file_list` *Split comma-separated file list*

Description

Parses a comma-separated string of file names into a character vector, trimming whitespace from each entry.

Usage

```
split_file_list(file_string)
```

Arguments

`file_string`	Comma-separated file names

Value

Character vector of individual file names

See Also

Other utilities: `get_diagram_themes()`, `is_valid_put_annotation()`

Examples

```
split_file_list("data.csv, results.rds, plot.png")
split_file_list("")
```

summary.putior_workflow *Summarize a putior workflow*

Description

Provides a structured summary of a putior workflow data frame.

Usage

```
## S3 method for class 'putior_workflow'
summary(object, ...)
```

Arguments

object	A putior_workflow object
...	Additional arguments (ignored)

Value

Invisibly returns a list with summary information

See Also

Other core-workflow: print.putior_theme(), print.putior_workflow(), put(), put_diagram(), put_theme()

Index

∗ **auto-annotation**
  put_auto, 11
  put_generate, 16
  put_merge, 18
∗ **core-workflow**
  print.putior_theme, 8
  print.putior_workflow, 8
  put, 9
  put_diagram, 13
  put_theme, 19
  summary.putior_workflow, 31
∗ **integration**
  putior_acp_manifest, 21
  putior_acp_server, 21
  putior_guide, 23
  putior_help, 24
  putior_mcp_server, 25
  putior_mcp_tools, 27
∗ **interactive**
  run_sandbox, 28
  set_putior_log_level, 29
∗ **language-support**
  ext_to_language, 2
  get_comment_prefix, 3
  get_detection_patterns, 4
  get_supported_extensions, 6
  list_supported_languages, 7
∗ **utilities**
  get_diagram_themes, 5
  is_valid_put_annotation, 6
  split_file_list, 30

ext_to_language, 2, *3, 4*, *6, 7*

get_comment_prefix, 3, *3, 4*, *6, 7*
get_detection_patterns, *3*, 4, *6, 7*
get_diagram_themes, 5, *7*, *20*, *30*
get_supported_extensions, *3, 4*, 6, *7*

is_valid_put_annotation, *5*, 6, *30*

list_supported_languages, *3, 4*, *6*, 7

print.putior_theme, 8, *9*, *11*, *15*, *20*, *31*
print.putior_workflow, 8, *8*, *11*, *15*, *20*, *31*
put, *8*, 9, *9*, *12–15*, *17*, *19, 20*, *29*, *31*
put_auto, 11, *17*, *19*
put_diagram, *8, 9*, *11*, 13, *13*, *19, 20*, *29*, *31*
put_generate, *13*, 16, *19*
put_merge, *13*, *17*, 18
put_theme, *8, 9*, *11*, *15*, 19, *31*
putior_acp_manifest, 21, *23–25*, *27, 28*
putior_acp_server, 21, *21*, *24, 25*, *27, 28*
putior_guide, *21*, 23, *23*, *25*, *27, 28*
putior_help, *21*, *23*, 24, *24*, *27, 28*
putior_mcp_server, *21*, *23, 24*, 25, *25*, *27, 28*
putior_mcp_tools, *21*, *23–25*, 27, *27*

run_sandbox, 28, *29*

set_putior_log_level, *9*, *12*, *15*, *17, 18*, 29, *29*
split_file_list, *5*, *7*, 30
summary.putior_workflow, *8, 9*, *11*, *15*, *20*, 31