Package 'SIAtools'

June 20, 2024

Title 'ShinyItemAnalysis' Modules Development Toolkit

Version 0.1.1

Description A comprehensive suite of functions designed for constructing and managing 'ShinyItemAnalysis' modules, supplemented with detailed guides, ready-to-use templates, linters, and tests. This package allows developers to seamlessly create and integrate one or more modules into their existing packages or to start a new module project from scratch.

License GPL (>= 3)

URL https://applstat.github.io/SIAtools/

Depends R (>= 3.6.0)

Imports cli, desc, fs, magrittr, purrr, rlang, shiny, usethis, yaml

Suggests glue, knitr, lintr, pkgload, rmarkdown, roxygen2, rstudioapi, spelling, testthat (>= 3.0.0), tibble, tidyr, withr, xml2

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.3.1

NeedsCompilation no

Author Jan Netik [cre, aut] (<https://orcid.org/0000-0002-3888-3203>), Patricia Martinkova [aut] (<https://orcid.org/0000-0003-4754-8543>)

Maintainer Jan Netik <netik@cs.cas.cz>

Repository CRAN

Date/Publication 2024-06-20 21:30:02 UTC

Contents

`add_module` *Add a new SIA module to your package*

Description

This is the workhorse of `{SIAtools}` package. The function checks if the package is properly configured for SIA modules and provides immediate fixes as needed. `add_module()` automatically puts a correct entry in SIA Modules Manifest of your package (which is created if not already present), and prepares `.R` file with the code template. Both files are automatically opened for you by default.

Usage

```
add_module(
  name = "new_module",
  title = NULL,
  category = NULL,
  open = TRUE,
  prefix = "sm_",
  proj = curr_proj()
)
```

Arguments

`name`	*character*, a name for the new SIA module.
`title`	*character*, new module's title. You can leave the default `NULL` and set manually in the manifest later on.
`category`	*character*, new module's category. The category dictates the tab within the `{ShinyItemAnalysis}` app to which the module should be appended. You can leave the default `NULL` and set manually in the manifest later on. Check the available categories using `list_categories()`.

open	Whether to open the manifest and module's source for interactive editing. Defaults to TRUE.
prefix	*character*, a prefix to denote SIA module. It's highly recommended to stick with the default "sm_" (standing for **S**IA **M**odule).
proj	*character*, a path to the project. Defaults to current project.

Value

No return value. Called for the side effects.

See Also

Other module management functions: `get_modules()`, `preview_module()`, `remove_module()`

Examples

```
## Not run:
# add the module called "test" and edit the details later on in the YAML
add_module("test")

# specify the title and category at creation time
add_module("test", title = "Test module", category = "Validity")

## End(Not run)
```

---

create_module_project *Create a new RStudio project prepared for SIA modules*

Description

The function is designed to be used primarily by RStudio "New Project Wizard". Create a new project by navigating through File > New Project > New Directory > ShinyItemAnalysis Module Project. See RStudio User Guide for the details.

Usage

```
create_module_project(path, ..., open = TRUE)
```

Arguments

path	*character*, a path to the new module. Use "../my_new_module" to create a module in the parent of the current working directory.
...	*used by RStudio only*
open	*logical*, whether to open the project in new RStudio session after creation. Defaults to TRUE.

Value

No return value. Called for the side effect.

Examples

```
if (interactive()) {
# create a new SIA module project in the parent of your working directory
create_module_project("../my_new_module")
}
```

---

curr_proj *Get a current project path*

Description

This is a thin wrapper around `usethis::proj_get()` that silences any messages.

Usage

```
curr_proj()
```

Value

The path to the current project.

See Also

Other helpers: `default_shiny_io_functions`, `edit_rstudio_shortcuts()`, `list_categories()`, `open_sm_manifest()`, `print.sm_manifest()`, `sia_head_tag()`

Examples

```
## Not run:
curr_proj()

## End(Not run)
```

default_shiny_io_functions *Default* {shiny} *input/output UI functions consulted by* module_namespace_linter()

Description

A character vector of function names whose calls are inspected for ns() omission. Name of each element denotes the original package. Please refer to module_namespace_linter() for more details.

Usage

```
default_shiny_io_functions
```

Format

An object of class character of length 25.

Details

Following functions are covered:

- shiny::actionButton
- shiny::actionLink
- shiny::checkboxGroupInput
- shiny::checkboxInput
- shiny::dateInput
- shiny::dateRangeInput
- shiny::fileInput
- shiny::numericInput
- shiny::passwordInput
- shiny::radioButtons
- shiny::selectInput
- shiny::sliderInput
- shiny::textAreaInput
- shiny::textInput
- shiny::varSelectInput
- shiny::dataTableOutput
- shiny::tableOutput
- shiny::uiOutput
- shiny::htmlOutput

- shiny::verbatimTextOutput
- shiny::imageOutput
- shiny::textOutput
- shiny::plotOutput
- plotly::plotlyOutput
- DT::DTOutput

See Also

Other helpers: curr_proj(), edit_rstudio_shortcuts(), list_categories(), open_sm_manifest(), print.sm_manifest(), sia_head_tag()

---

edit_rstudio_shortcuts

Show RStudio Keyboard Shortcuts

Description

Shows a popup window with RStudio keyboard shortcuts. Applicable only in RStudio and in interactive R session.

Usage

```
edit_rstudio_shortcuts()
```

Details

You can quickly reach out solicited addin function by typing it in the `Filter...` box in the very top of the window. Then double click at the blank space just next to the addin function name and press down desired key or key combination. Apply the changes and from now on, just call the addin function with one keystroke. For more details, navigate to RStudio documentation.

Value

No return value. Called for side effect.

See Also

Other helpers: curr_proj(), default_shiny_io_functions, list_categories(), open_sm_manifest(), print.sm_manifest(), sia_head_tag()

Examples

```
if (interactive()) {
edit_rstudio_shortcuts()
}
```

get_modules *Get the SIA Modules Manifest for the currently developed package*

Description

Returns a list with all modules for the current package as described in its SIA Modules Manifest, which resides at `/inst/sia/modules.yml` and is generated with `add_module()` calls. Can be formatted as a `tibble` using the respective print method.

Usage

```
get_modules(proj = curr_proj())
```

Arguments

`proj`	*character*, a path to the project. Defaults to current project.

Value

A SIA Modules Manifest of class `sm_manifest`. Inherits from a `list`.

See Also

Other module management functions: `add_module()`, `preview_module()`, `remove_module()`

Examples

```
## Not run:
get_modules()

## End(Not run)
```

lint_ns *Check the* `{shiny}` *module UI functions for* `ns()` *omission*

Description

This is a simple wrapper of `lintr::lint_package()` call using only the `module_namespace_linter()` from `{SIAtools}`. See the linter documentation for more details.

Usage

```
lint_ns(path = curr_proj(), ...)
```

Arguments

`path`	*character*, path to the package root directory. Default is the current project's directory.
`...`	Arguments passed on to `lintr::lint_package` `parse_settings` Logical, default `TRUE`. Whether to try and parse the settings; otherwise, the `default_settings()` are used. `relative_path` if `TRUE`, file paths are printed using their path relative to the base directory. If `FALSE`, use the full absolute path. `exclusions` exclusions for `exclude()`, relative to the package path. `show_progress` Logical controlling whether to show linting progress with a simple text progress bar *via* `utils::txtProgressBar()`. The default behavior is to show progress in `interactive()` sessions not running a testthat suite.

Value

An object of class c("lints", "list"), each element of which is a "list" object.

See Also

Other linter-related functions: `module_namespace_linter()`

Examples

```
## Not run:
lint_ns()

## End(Not run)
```

`list_categories` *List the available SIA module categories*

Description

Lists all available categories a SIA module can be placed in. SIA app will place the module with illegal category under "Modules".

Usage

```
list_categories()
```

Value

A character vector.

See Also

Other helpers: curr_proj(), default_shiny_io_functions, edit_rstudio_shortcuts(), open_sm_manifest(), print.sm_manifest(), sia_head_tag()

Examples

```
list_categories()
```

---

module_namespace_linter *Require usage of* ns() *in* inputId *and* outputId *arguments of UI functions in* {shiny} *modules*

Description

A custom *linter* to be used by {lintr} package. Checks the functions in a module's UI part for any missing ns() calls. These are often omitted when working with the plain {shiny} or SIA modules. More details follows below.

Usage

```
module_namespace_linter(
  io_funs = default_shiny_io_functions,
  io_args = c("inputId", "outputId"),
  ns_funs = c("ns", "NS")
)
```

Arguments

io_funs	*character*, {shiny} input/output UI functions to check. Defaults to default_shiny_io_functions, which covers all native ones and several others from {plotly} or {DT}. The functions must include the namespace, i.e., shiny::textInput.
io_args	*character*, arguments of UI functions to check. inputId and outputId by default. These are checked even if unnamed. Named arguments that partially match are ignored and discouraged.
ns_funs	*character*, function names that are considered valid in order to "namespace" inputs' or outputs' IDs. Defaults to both ns and NS (although we recommend to stick with the former, which is predefined in the module template).

Details

How to use this linter:

The easiest way is to call lint_ns() which is essentially a wrapper around:

```
lintr::lint_package(linters = module_namespace_linter())
```

Both calls use our linter for the whole package. However, note that *only* `module_namespace_linter` is considered. Using this custom linter with the native ones is somewhat complicated, but not impossible. To the best of our knowledge, the only place where the {lintr} documentation mentions the actual usage of external linters, is in linters_with_tags() help page. According to that, you can pass the following call to `linters` argument in any supported `lintr::lint_*` function:

```
lintr::linters_with_tags(
 tags = NULL, packages = c("lintr", "SIAtools")
)
```

That should select all linters available in both packages.

It is also possible to set up a configuration file that enables you to shorten calls to {lintr} functions, use RStudio Addins to lint an active file, or even apply linters during continuous integration workflows, e.g., in GitHub Actions or in Travis. To opt for that, create `.lintr` file at your package's root and fill in the following line:

```
linters: linters_with_tags(tags = NULL, packages = "SIAtools")
```

Then, you can use the provided addins or call `lintr::lint_package()` to get your modules checked.

What the linter does:

By default, the linter looks for any `inputId` or `outputId` arguments of {shiny}'s UI functions (such as numericInput or plotOutput, respectively), and tests if the values assigned to the arguments are all "namespaced", i.e., wrapped in `ns()` function. This is crucial for inputs and outputs in the UI portion of a module to match their counterparts in the server logic chunk.

Only {shiny} UI calls that are inside of a tagList in a function ("lambda" shorthand, `\( )`, applies as well) are inspected. This is because we don't want to cause false alarms for any "ordinary" {shiny} apps that aren't modules. All UI portions of modules are usually defined as functions, and all input/output UI functions are inside a tagList, so we opted for the this strategy to minimalize false positive matches outside {shiny} modules.

We look for any `inputId` or `outputId` arguments that are named as such. On top of that, the `ns()` omission is detected even if you call the function without named arguments that would be evaluated as input or output IDs. However, if you use partial matching (`numericInput(inp = "input")`), the actual input won't get linted, even though it should, as it is eventually evaluated as `inputId`. The same applies for arguments defined outside the call and passed as a variable, e.g., `inp <- "input"; numericInput(inputId = inp)`. That is tricky to catch in a static code analysis, which is employed in this linter.

Value

A `linter` closure. To be used by {lintr} only. See the first example below.

See Also

linters for a complete list of linters available in lintr.

Other linter-related functions: `lint_ns()`

Examples

```
# will produce lints
lintr::lint(
  text =
    "module_ui <- function(id, imports, ...) {
      tagList(
        numericInput(inputId = \"input_id_without_ns\", ...)
      )
    }",
  linter = module_namespace_linter()
)

# is OK
lintr::lint(
  text =
    "module_ui <- function(id, imports, ...) {
      tagList(
        numericInput(inputId = ns(\"input_id_with_ns\"), ...)
      )
    }",
  linter = module_namespace_linter()
)
```

`open_sm_manifest` *Open SIA Modules Manifest for Editing*

Description

Open SIA Modules Manifest for Editing

Usage

```
open_sm_manifest(proj = curr_proj())
```

Arguments

`proj`	*character*, a path to the project. Defaults to current project.

Value

character, a path to SIA Module Manifest (invisibly).

See Also

Other helpers: `curr_proj()`, `default_shiny_io_functions`, `edit_rstudio_shortcuts()`, `list_categories()`, `print.sm_manifest()`, `sia_head_tag()`

Examples

```
## Not run:
open_sm_manifest()

## End(Not run)
```

`preview_module` *Preview a module*

Description

Previews a SIA module in a standalone development environment. See the details below.

Usage

```
preview_module(
  module_id = NULL,
  ui_imports = NULL,
  server_imports = NULL,
  ui_elements = sia_head_tag(),
  save_and_document = TRUE,
  load = TRUE,
  proj = curr_proj(),
  ...
)
```

Arguments

`module_id`	*character*, name of the module to preview (including the prefix). If `NULL` (the default), all modules discovered by `get_modules()` are listed and you are asked to pick one.
`ui_imports`	*list*, UI objects exported from the `{ShinyItemAnalysis}` app. *Not used at the moment*.
`server_imports`	*list*, reactive objects exported from the `{ShinyItemAnalysis}` app. See the Details.
`ui_elements`	elements to include in `fluidPage`, preferably packed in `tagList()`.
`save_and_document`	*logical*, whether to save all unsaved files (only available in RStudio) and document the package. Defaults to `TRUE`. Note that documenting the package is necessary if you use any functions from external packages (to produce the `NAMESPACE`).
`load`	*logical*, whether to load your package before running the module preview. Defaults to `TRUE`. Note that you have to load the package by yourself or install it in the usual way if you set this to `FALSE`.
`proj`	*character*, a path to the project. Defaults to current project.

... Arguments passed on to shiny::shinyApp

onStart A function that will be called before the app is actually run. This is only needed for shinyAppObj, since in the shinyAppDir case, a global.R file can be used for this purpose.

options Named options that should be passed to the runApp call (these can be any of the following: "port", "launch.browser", "host", "quiet", "display.mode" and "test.mode"). You can also specify width and height parameters which provide a hint to the embedding environment about the ideal height/width for the app.

uiPattern A regular expression that will be applied to each GET request to determine whether the ui should be used to handle the request. Note that the entire request path must match the regular expression in order for the match to be considered successful.

enableBookmarking Can be one of "url", "server", or "disable". The default value, NULL, will respect the setting from any previous calls to enableBookmarking(). See enableBookmarking() for more information on bookmarking your app.

Details

The function takes module's function bindings and puts (or evaluates) them inside a bare bone shiny::shinyApp(). By default, a customized head tag is injected in order to mimic the "environment" of full {ShinyItemAnalysis} app. See sia_head_tag() for more details. Besides, a onSessionEnded hook is set to call shiny::stopApp() after the client disconnects, so the "process" is automatically quit after you close the preview in your browser or RStudio viewer.

In order to use the function bindings, preview_module() attempts to load your package without the actual installation by default. **Note that you have to install the package as usual for {ShinyItemAnalysis} to detect your modules.**

Using objects from the {ShinyItemAnalysis} app:

Note that this "emulated" preview environment is really meant to test the basic UI layout and functionality and is not able to receive any object from {ShinyItemAnalysis} app. However, you can pass any object like {ShinyItemAnalysis} does to server_imports argument manually. For further details and examples, please refer to vignette("imports", "SIAtools") vignette.

Value

Shiny app object of class shiny.appobj.

See Also

Other module management functions: add_module(), get_modules(), remove_module()

Examples

```
if (interactive()) {
  preview_module()
}
```

print.sm_manifest *Print a SIA Modules Manifest*

Description

Prints out the SIA Modules Manifest obtained through `get_modules()`. By default, a plain YAML content is returned, but you can also get a formatted output in a `tibble`, which is suitable for packages with large number of SIA modules.

Usage

```
## S3 method for class 'sm_manifest'
print(x, ..., as_tibble = FALSE)
```

Arguments

`x`	*sm_manifest* object, i.e., an output of `get_modules()`.
`...`	Not used at the moment.
`as_tibble`	*logical*, print the manifest as a `tibble`? Defaults to `FALSE`.

Value

Called for side effects by default. Returns a `tibble` if `as_tibble` argument is set to `TRUE`.

See Also

Other helpers: `curr_proj()`, `default_shiny_io_functions`, `edit_rstudio_shortcuts()`, `list_categories()`, `open_sm_manifest()`, `sia_head_tag()`

remove_module *Remove a module*

Description

Removes the given module from the SIA Module Manifest and deletes the respective `.R` file.

Usage

```
remove_module(module_id = NULL, proj = curr_proj())
```

Arguments

`module_id`	*character*, name of the module to remove (including the prefix). If `NULL` (the default), all modules discovered by `get_modules()` are listed and you are asked to pick one.
`proj`	*character*, a path to the project. Defaults to current project.

Value

No return value. Called for the side effect.

See Also

Other module management functions: `add_module()`, `get_modules()`, `preview_module()`

Examples

```
## Not run:
remove_module()

## End(Not run)
```

`sia_head_tag` *Core HTML head tag from ShinyItemAnalysis*

Description

This function is intended to be used for SIA module preview only, which is done by default in `preview_module()`. It provides a HTML head tag with a math rendering facility provided by the *KaTeX* library, and some CSS rules to format tables. *If printed in the console, expect nothing to be shown.*

Usage

```
sia_head_tag()
```

Value

HTML head tag of class `shiny.tag`.

See Also

Other helpers: `curr_proj()`, `default_shiny_io_functions`, `edit_rstudio_shortcuts()`, `list_categories()`, `open_sm_manifest()`, `print.sm_manifest()`

Index