# Package: mighty.component (via r-universe)

June 1, 2026

**Title** Mighty Components for ADaM Generation

**Version** 0.1.0.9000

**Description** Components to be used inside the 'mighty' framework for generation of ADaM programs.

**License** Apache License (>= 2)

**URL** https://novonordisk-opensource.github.io/mighty.component, https://github.com/NovoNordisk-OpenSource/mighty.component

**Depends** R (>= 4.1)

**Imports** cli, glue, R6 (>= 2.4.0), rlang, whisker, xml2, xmlparsedata, zephyr

**Suggests** admiral, callr, covr, dplyr, gh, jsonlite, knitr, pharmaverseadam, remotes, rmarkdown, testthat (>= 3.0.0), tibble, tidyr, withr

**VignetteBuilder** knitr

**Config/roxygen2/version** 8.0.0

**Config/testthat/edition** 3

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**Config/pak/sysreqs** libxml2-dev

**Repository** https://novonordisk-opensource.r-universe.dev

**Date/Publication** 2026-06-01 10:33:36 UTC

**RemoteUrl** https://github.com/NovoNordisk-OpenSource/mighty.component

**RemoteRef** HEAD

**RemoteSha** b173119250caf970c14032b54e9a3d798edcba13

## Contents

---

`get_component` *Retrieve mighty code component*

---

## Description

Retrieve a mighty code component.

- `get_component()`: Returns an object of class `mighty_component`.
- `get_rendered_component()`: Returns an object of class `mighty_component_rendered`.

When rendering a component the required list of parameters depends on the individual component. Check the documentation of the local component for details.

## Usage

```
get_component(component, repos = NULL)

get_rendered_component(component, params = list(), repos = NULL)
```

## Arguments

| | |
|---|---|
| `component` | `character` path to a component file (`.R` or `.mustache`). |
| `repos` | prioritised `character` vector of locations to look for component in. See details. |
| `params` | named `list` of input parameters. Passed along to `mighty_component$render()`. |

## Details

Processes different component types based on file extension:

- `.R`: Extracts and renders custom functions.
- `.mustache`: Creates components from the template files.

The `repos` parameter accepts a character vector of locations to search, in priority order. Each element is either a local directory path or a GitHub source in `owner/repo`, `owner/repo/subdir`, or `owner/repo@ref` format. The first match is returned. Defaults to the current directory.

## See Also

mighty_component, mighty_component_rendered

## Examples

```
path <- system.file("examples", "ady.mustache", package = "mighty.component")
get_component(path)
```

---

`get_test_component` *Create a testable component for unit testing*

---

## Description

Creates a `mighty_component_test` object from a rendered component, enabling structured unit testing with optional coverage checking.

See mighty_component_test for a description of the testing workflow.

## Usage

```
get_test_component(
  component,
  params = list(),
  repos = NULL,
  check_coverage = TRUE,
  teardown_env = parent.frame()
)
```

## Arguments

| | |
|---|---|
| `component` | `character` path to a component file (`.R` or `.mustache`). |
| `params` | named `list` of input parameters. Passed along to `mighty_component$render()`. |
| `repos` | prioritised `character` vector of locations to look for component in. See details. |
| `check_coverage` | `logical(1)` Whether to automatically check test coverage when the test completes. If `TRUE` (default), coverage is verified via `test_component$check_coverage()` in a deferred call. |
| `teardown_env` | The environment in which to register the deferred coverage check. Defaults to the caller's environment (`parent.frame()`). This controls when `check_coverage()` executes during test teardown. |

## Value

A `mighty_component_test` object.

## See Also

`get_rendered_component()`, mighty_component_test

`list_components` *List components in directories*

## Description

List all available mighty components (`.R` and `.mustache` files) in the specified directories.

## Usage

```
list_components(path, as = c("character", "list", "tibble"))
```

## Arguments

| | |
|---|---|
| `path` | `character` vector of directory paths to search for components. |
| `as` | Format to list the components in. Default `character` lists component IDs (filenames without extension), while `list` and `tibble` show detailed component metadata. |

## Value

Depending on `as`:

- `character`: vector of component IDs
- `list`: list of component metadata (id, title, description, params, depends, outputs, code)
- `tibble`: tibble with one row per component

## See Also

`get_component()`

## Examples

```
path <- system.file("examples", package = "mighty.component")
list_components(path)

list_components(path, as = "list") |> str(max.level = 1)
```

---

`mighty.component-options` *Options for mighty.component*

---

## Description

**verbosity_level:**

Verbosity level for functions in mighty.component. See zephyr::verbosity_level for details.

- Default: `NA_character_`
- Option: `mighty.component.verbosity_level`
- Environment: `R_MIGHTY.COMPONENT_VERBOSITY_LEVEL`

---

`mighty_component` *Mighty component*

---

## Description

Class for a generic mighty component.

In the mighty framework, a "component" is a code template that processes input data and returns a modified version with new columns or rows. Mighty components share a common structure and roxygen-like documentation pattern, facilitating their use inside mighty.

## Details

Templates are `character` vectors of R code that are interpreted. Dynamic use of variables etc. are supported using the mustache framework. Dynamic parameters are specified using `{{ variable_name }}`.

**Documentation:**

A template is required to be documented with the following tags similar to when documenting functions using roxygen2:

| Tag | Description | Example |
|---|---|---|
| `@title` | Title of the component | `@title My component` |
| `@description` | Description of the component | `@description text text` |
| `@param` | Specifies input used to render the component | `@param variable new var` |
| `@type` | Specifies type: column, row, parameter, internal | `@type column` |
| `@origin` | CDISC origin (optional) | `@origin Derived` |
| `@depends` | Required input variable (repeat if several) | `@depends {{ domain }} USUBJID` |
| `@outputs` | Variables created (repeat if several) | `@outputs NEWVAR` |
| `@code` | Everything under this tag defines the component code | `@code` |

**Conventions:**

A component template follows these conventions:

1. The input data set is always called `{{ domain }}`.
2. Additional parameters used to render the template into R code are documented with the `@param` tag.
3. The template ends with creating a modified version of `{{ domain }}`.
4. Template documented with the roxygen-like tags above

**Example:**

Below is an example of a mighty component template that creates a new dynamic variable `variable` as twice the value of the dynamic input `x`, that should already by in the input data set `{{ domain }}`.

```
#' @title Title for my component
#' @description
#' A more in depth description of what is being done
#'
#' @param variable dynamic output if applicable
#' @param x some other input to the component
#' @type column
#' @origin Derived
#' @depends {{ domain }} {{ x }}
#' @outputs {{ variable }}
#' @code
{{ domain }} <- {{ domain }} |>
  dplyr::mutate(
    {{ variable }} = 2 * {{ x }}
  )
```

When rendered with parameters `variable = "A"` and `x = "B"` the rendered code used in mighty becomes:

```
{{ domain }} <- {{ domain }} |>
  dplyr::mutate(
    A = 2 * B
  )
```

## Active bindings

`id` Component ID.

`title` Title for the component.

`description` Description of the component.

`code` The code block of the component.

`template` The complete template.

`type` The type of the component. Can be one of column, row, parameter, internal.

`origin` CDISC origin. One of Assigned, Collected, Derived, Not Available, Other, Predecessor, Protocol or `NULL`.

`depends` Data.frame listing all the components dependencies.

`outputs` List of the new columns created by the component.

`params` Data.frame listing parameters that need to be supplied when rendering the component.

## Methods

### Public methods:

- `mighty_component$new()`
- `mighty_component$print()`
- `mighty_component$render()`
- `mighty_component$document()`
- `mighty_component$clone()`

`mighty_component$new()`: Create component from template.

*Usage:*

```
mighty_component$new(template, id)
```

*Arguments:*

`template` `character` template code. See details for how to format.

`id` `character` ID of the component.

`mighty_component$print()`: Print method displaying the component information.

*Usage:*

```
mighty_component$print()
```

*Returns:* (`invisible`) self

`mighty_component$render()`: Render component with supplied values. Supports mustache templates and uses `whisker::whisker.render()`.

*Usage:*

```
mighty_component$render(...)
```

*Arguments:*

`...` Parameters used to render the template. Must be named, and depends on the template.

*Returns:* Object of class mighty_component_rendered

`mighty_component$document()`: Create standard documentation in markdown format.

*Usage:*

```
mighty_component$document()
```

`mighty_component$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
mighty_component$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## See Also

`get_component()`, mighty_component_rendered

mighty_component_rendered *Rendered mighty component*

## Description

Class for a rendered mighty component.

Once rendered a component can be used to:

- Stream into an R script
- Evaluate the generated code in an environment

## Super class

mighty_component -> mighty_component_rendered

## Methods

### Public methods:

- mighty_component_rendered$new()
- mighty_component_rendered$print()
- mighty_component_rendered$stream()
- mighty_component_rendered$eval()
- mighty_component_rendered$clone()

**mighty_component_rendered$new():** Create component from rendered template.

*Usage:*

```
mighty_component_rendered$new(template, id)
```

*Arguments:*

`template` character Rendered template such as output from `mighty_component$render()`.

`id` character ID of the component.

**mighty_component_rendered$print():** Print rendered component

*Usage:*

```
mighty_component_rendered$print()
```

*Returns:* (invisible) self

**mighty_component_rendered$stream():** Stream rendered code into a script (appended)

*Usage:*

```
mighty_component_rendered$stream(path)
```

*Arguments:*

`path` character(1) path to the R script to stream code into.

`mighty_component_rendered$eval()`: Evaluate code in a specified environment.

*Usage:*

```
mighty_component_rendered$eval(envir = parent.frame())
```

*Arguments:*

`envir` Environment to evaluate in. Parsed to `eval()`. Defaults to using the current environment with `parent.frame()`.

`mighty_component_rendered$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
mighty_component_rendered$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## See Also

`get_rendered_component()`

---

`mighty_component_test` *Test mighty component*

---

## Description

Class for unit testing a mighty component with code coverage tracking. Runs component code in an isolated R session and tracks which lines are executed during testing.

## Details

Always use `get_test_component()` to create instances for testing. The test workflow is:

1. Create test component with `get_test_component()`
2. Assign input data with `$assign()`
3. Execute and track coverage with `$eval()`
4. Retrieve results with `$get()`
5. Test results with `expect_*()` functions from `{testthat}`

Coverage is automatically checked at test teardown via `$check_coverage()`.

## Super classes

`mighty_component` -> `mighty_component_rendered` -> `mighty_component_test`

## Active bindings

`percent_coverage` `numeric` Percentage of lines covered (0-100).

`line_coverage` `data.frame` with columns `line` and `value` showing execution count per line.

## Methods

### Public methods:

- mighty_component_test$new()
- mighty_component_test$print()
- mighty_component_test$assign()
- mighty_component_test$get()
- mighty_component_test$ls()
- mighty_component_test$eval()
- mighty_component_test$check_coverage()
- mighty_component_test$clone()

`mighty_component_test$new()`: Create test component from rendered template.

*Usage:*

```
mighty_component_test$new(template, id)
```

*Arguments:*

`template` `character` Rendered template such as output from `mighty_component$render()`.
`id` `character` ID of the component.

`mighty_component_test$print()`: Print method showing component and test coverage

*Usage:*

```
mighty_component_test$print()
```

`mighty_component_test$assign()`: Assign a variable in the isolated test session.

*Usage:*

```
mighty_component_test$assign(x, value)
```

*Arguments:*

`x` `character` Name of the variable to assign.
`value` Value to assign to the variable.

*Returns:* `self` invisibly, for method chaining.

`mighty_component_test$get()`: Retrieve a variable from the isolated test session.

*Usage:*

```
mighty_component_test$get(x)
```

*Arguments:*

`x` `character` Name of the variable to retrieve.

*Returns:* The value of the variable.

`mighty_component_test$ls()`: List all objects in the isolated test session.

*Usage:*

```
mighty_component_test$ls()
```

*Returns:* `character` vector of variable names.

`mighty_component_test$eval()`: Execute the component code and update coverage tracking.

*Usage:*

```
mighty_component_test$eval()
```

*Returns:* `self` invisibly, for method chaining.

`mighty_component_test$check_coverage()`: Check that all lines in the component were executed at least once. Throws an error if any lines have zero coverage.

*Usage:*

```
mighty_component_test$check_coverage()
```

*Returns:* `self` invisibly if all lines are covered.

`mighty_component_test$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
mighty_component_test$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

# Index