

# Package: mighty (via r-universe)

October 19, 2026

**Title** ADaM Program Generation Through Declarative Specifications

**Version** 0.0.0.9042

**Description** Provides tools for generating ADaM programs in clinical trials using a declarative approach. Users specify what data structure and derivations they need through YAML-based specifications, rather than writing imperative code. The package handles dependency tracking between columns and rows, orchestrates code components, and automatically generates executable R programs for ADaM dataset creation.

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

**URL** <https://novonordisk-opensource.github.io/mighty/>

**Depends** R (>= 4.1)

**Imports** assertthat, checkmate, cli, data.table, dplyr, glue, igraph, mighty.component (>= 0.1.0.9005), mighty.metadata (>= 0.1.0.9006), purrr, R6, RColorBrewer, rlang, S7, S7schema, styler, utils, visNetwork, zephyr

**Suggests** admiral, arrow, connector, gh, knitr, labelled, pharmaverseadam, pharmaversesdtm (>= 0.2.0), quarto, remotes, rmarkdown, roxygen2, testthat (>= 3.0.0), tools, whisker, withr, yaml

**VignetteBuilder** knitr, quarto

**Remotes** NovoNordisk-OpenSource/mighty.component,  
NovoNordisk-OpenSource/mighty.metadata

**Config/roxygen2/version** 8.1.1

**Config/testthat/edition** 3

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

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

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

**Date/Publication** 2026-10-19 08:17:12 UTC

**RemoteUrl** <https://github.com/NovoNordisk-OpenSource/mighty>  
**RemoteRef** HEAD  
**RemoteSha** 7953dd03f9c5086a195692735b90df75d8ab3382

Contents

|                               |          |
|-------------------------------|----------|
| generate_adam_code . . . . .  | 2        |
| write_adam_programs . . . . . | 5        |
| <b>Index</b>                  | <b>6</b> |

---

|                    |                                               |
|--------------------|-----------------------------------------------|
| generate_adam_code | <i>Generate Complete Set of ADaM Programs</i> |
|--------------------|-----------------------------------------------|

---

Description

This function orchestrates the complete workflow for generating ADaM (Analysis Data Model) programs from ADaM specifications and trial metadata. It processes specifications, validates dependencies, creates action sequences, and renders executable R code for ADaM dataset creation.

Usage

```
generate_adam_code(  
  adam_specifications,  
  path_connector_config,  
  check_cross_domain_adam_dependencies = TRUE,  
  data_context = NULL  
)
```

Arguments

- adam\_specifications  
Character string. Directory path containing ADaM specification YAML files (one per domain) and optionally a `_mighty.yml` framework configuration file with external dataset definitions and keys.
- path\_connector\_config  
Character string. File path to the connector configuration file (e.g., `"_connector.yml"`). This path is inserted exactly as written into the generated programs (no validation or transformation is performed). The generated programs use this path to connect to data when they execute - mighty itself does not read or validate the connector configuration file. Prefix with `!expr` to embed an R expression that is evaluated at runtime by the generated program (e.g., `'!expr here::here("_connector.yml")'`).
- check\_cross\_domain\_adam\_dependencies  
Logical. If TRUE (default), validates dependencies across different ADaM domains. If FALSE, only validates dependencies within individual domains.

**data\_context** Optional list or environment providing additional context about available data sources for executable program generation. If NULL, all programs are considered potentially executable.

## Details

The function executes the following workflow:

1. Reads and validates ADaM specifications and trial metadata
2. Sets up initial action configurations with dependency validation
3. Adds domain initialization, filtering, and data reading actions
4. Creates dependency graph and organizes actions in execution order
5. Adds data writing actions and checks executable status
6. Renders R code for both complete and executable program sets
7. Compiles individual actions into complete, runnable programs

Each generated program includes all necessary data transformations, derivations, and output operations for creating a specific ADaM dataset according to the provided specifications.

## Value

A named list containing the complete ADaM program generation results:

**programs** Named list of complete R programs, one per ADaM domain. Each element is a character string containing a fully executable R script. Programs include all dependency-ordered actions and are ready to execute independently. Names are prefixed with the required execution order.

**program\_sequence** *For debugging only.* Data.table containing the complete action sequence with fully rendered R code for all programs. Each row represents a single action with action metadata. Actions are ordered by dependency requirements within each domain. This provides a detailed view of the execution plan before compilation into complete programs.

**executable\_programs** Named list of programs that can be executed with the currently available data sources (as determined by data\_context). Structure identical to programs but includes only code where all required input data is available. If data\_context is NULL, this will match programs. Used to identify which ADaM derivations can be generated given the current data availability.

**executable\_program\_sequence** *For debugging only.* Data.table containing the action sequence for executable programs only. Structure similar to program\_sequence but filtered to include only actions from code that can be executed with available data. Provides visibility into which specific transformations will run when executing the available programs.

**edges** *For debugging only.* Data.table defining the dependency graph between actions. Contains columns parent\_node and node\_id, representing directed edges where parent actions must execute before child actions. Edges are created from both column dependencies (when one action produces a column another action consumes) and row dependencies (explicit row-level operations). Includes synthetic edges connecting actions with no dependencies to domain initialization actions. Self-referential edges are removed. Used for debugging action execution order and dependency resolution.

**actions** *For debugging only.* Data.table containing base action configurations before code rendering and program organization. Each row represents a single action with columns: node\_id (unique action identifier), domain (ADaM domain name), code\_id (reference to code component or NA), type (action type: col\_copy, col\_rename, col\_mutate, col\_echo, col\_compute, row\_compute, init\_domain, or filter\_domain), outputs (list column of character vectors showing columns produced), depend\_cols (nested data.table with column\_name, domain, and domain\_type showing column dependencies), depend\_rows (list of node\_ids this action depends on for row operations), and parameters (named list of user-provided parameters). Reflects the internal action data model before execution ordering. Used for debugging action setup and dependency validation.

**rendered\_components** *For debugging only.* Named list of code components that were successfully rendered using Mustache templates during the generation process. Each element contains the rendered R code for a specific component (e.g., derivation functions). Component names correspond to code\_id values referenced in the action specifications. Used for inspecting how template parameters were resolved and for debugging component rendering.

## File Requirements

- ADaM specification file must be a valid YAML file with ADaM specifications following the schema defined in mighty.metadata
- Trial metadata file must contain valid study configuration
- Connector configuration file path must be valid for the generated programs to execute

## Error Handling

The function will stop execution if:

- ADaM specifications or trial metadata files cannot be read or are invalid
- Dependency validation fails (missing required columns)
- Trial configuration is malformed

## Examples

```
## Not run:
# Generate ADaM programs with full dependency checking
result <- generate_adam_code(
  adam_specifications = "path/to/yaml_specs_directory",
  path_connector_config = "path/to/trial_directory/_connector.yml",
  check_cross_domain_adam_dependencies = TRUE
)

# Access generated programs
adsl_program <- result$programs$ADSL
executable_programs <- result$executable_programs

## End(Not run)
```

---

|                     |                                    |
|---------------------|------------------------------------|
| write_adam_programs | <i>Write ADaM Programs to Disk</i> |
|---------------------|------------------------------------|

---

## Description

Writes generated ADaM programs to individual R files and optionally applies code styling.

## Usage

```
write_adam_programs(programs, dir, style = FALSE)
```

## Arguments

|          |                                                                                                                           |
|----------|---------------------------------------------------------------------------------------------------------------------------|
| programs | Named list. Each element contains program code as character vector, with names used as file names (without .R extension). |
| dir      | Character. Output directory path where R files will be written.                                                           |
| style    | Logical. Whether to apply code styling using <code>styler::style_file()</code> . Default is FALSE.                        |

## Details

For each program in the list:

- Creates filename by appending ".R" to the list element name
- Collapses code vector into single string with newline separators
- Writes code to file in specified directory
- Optionally applies styler formatting if requested

## Value

Logical. Returns TRUE invisibly upon successful completion.

## Examples

```
## Not run:
# Write programs without styling
programs <- list(
  ADSL = c("library(dplyr)", "ADSL <- DM |> select(USUBJID)"),
  ADAE = c("library(dplyr)", "ADAE <- AE |> filter(!is.na(AEDECOD))")
)

write_adam_programs(programs, dir = "output/adam")
# Creates: output/adam/ADSL.R, output/adam/ADAE.R

# Write with styling
write_adam_programs(programs, dir = "output/adam", style = TRUE)

## End(Not run)
```

# Index

`generate_adam_code`, [2](#)  
`styler::style_file()`, [5](#)  
`write_adam_programs`, [5](#)