

Package: odin2 (via r-universe)

October 6, 2026

Title Next generation odin

Version 0.3.46

Description Temporary package for rewriting odin.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

URL <https://mrc-ide.github.io/odin2>, <https://github.com/mrc-ide/odin2>

BugReports <https://github.com/mrc-ide/odin2/issues>

Imports cli, dust2 (>= 0.3.9), glue, monty (>= 0.3.30), rlang

Suggests decor, deSolve, fs, knitr, rmarkdown, mockery, numDeriv,
pkgload, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Remotes mrc-ide/dust2, mrc-ide/monty

VignetteBuilder knitr

Language en-GB

Repository <https://mrc-ide.r-universe.dev>

Date/Publication 2026-05-07 12:37:18 UTC

RemoteUrl <https://github.com/mrc-ide/odin2>

RemoteRef main

RemoteSha 94e7b52b0a4e50842129472ddd6d6090f649fd4a

Contents

odin	2
odin_error_explain	3
odin_migrate	4
odin_package	5
odin_show	7
odin_validate	8

Index**11**

odin	<i>Compile an odin model</i>
------	------------------------------

Description

Compile an odin model, yielding a `dust_system_generator` object.

Usage

```
odin(
  expr,
  ...,
  input_type = NULL,
  quiet = NULL,
  workdir = NULL,
  debug = NULL,
  skip_cache = FALSE,
  compatibility = NULL,
  check_bounds = NULL,
  target = NULL
)
```

Arguments

<code>expr</code>	Odin code as the path to a file (a string), a character vector of code, or as an expression (typically within braces <code>{}</code>).
<code>...</code>	Unused, must be empty. This allows us to add and reorder remaining arguments at will, as all arguments must be specified by name, not position).
<code>input_type</code>	An optional string describing the type of input for <code>expr</code> - must be one of <code>file</code> , <code>text</code> or <code>expression</code> . If given, this skips the type detection logic and odin will throw an error if the wrong type of input is given. Using this may be beneficial in programmatic environments.
<code>quiet</code>	Logical, indicating if compilation messages from <code>pkgbuild</code> should be displayed. Error messages will be displayed on compilation failure regardless of the value used. If <code>NULL</code> is given, then we take the value from <code>DUST_QUIET</code> if set, or <code>FALSE</code> otherwise.
<code>workdir</code>	Optional working directory to use. If <code>NULL</code> , the behaviour depends on the existence of the environment variable <code>DUST_WORKDIR_ROOT</code> . If it is unset we use a session-specific temporary directory (generated by <code>tempfile()</code>). If <code>DUST_WORKDIR_ROOT</code> is set, then we use a stable generated filename within this directory, which allows different sessions to effectively share a cache. If you pass a directory name here as a string, then we will use that directory to write all code, which allows you to inspect the generated code. See vignette("details") for more information.
<code>debug</code>	Passed to <code>pkgbuild::compile_dll</code> , this will build a debug library. If <code>NULL</code> is given, then we take the value from <code>DUST_DEBUG</code> if set, or <code>FALSE</code> otherwise.

skip_cache	Logical, indicating if the cache of previously compiled systems should be skipped. If TRUE then your system will not be looked for in the cache, nor will it be added to the cache after compilation.
compatibility	Compatibility mode to use. Valid options are "warning", which updates code that can be fixed, with warnings, and "error", which will error. The option "silent" will silently rewrite code, but this is not recommended for general use as eventually the compatibility mode will be removed (this option is primarily intended for comparing output of odin1 and odin2 models against old code). The default, NULL, currently corresponds to warning.
check_bounds	Control over static array bounds checking. This is enabled by default, but is prone to false positives, erroring where a read or write appears out of bounds but is actually ok. This argument exists to allow you to disable the check and compile the model anyway. Future versions may allow specific lines to be ignored, which will provide finer control and allow you to use the bits of the checks that are actually helpful. You can also pass TRUE here to mean "error" or FALSE to mean "disabled".
target	Compilation target. If provided, it must be one of cpp or js. In future, we hope to support cuda to enable support compilation to run on NVIDIA GPUs. The default, NULL, falls back on the option odin2.target, and if that is unset we currently select cpp, but this is subject to change once the JavaScript support is complete.

Value

A dust_system_generator object, suitable for using with dust functions (starting from [dust2::dust_system_create](#))

Examples

```
# A random walk:
gen <- odin({
  initial(x) <- 0
  update(x) <- Normal(x, 1)
})

sys <- dust2::dust_system_create(gen, list(), n_particles = 10)
y <- dust2::dust_system_simulate(sys, 0:100)
matplot(t(y[1, , ]), type = "l", lty = 1, xlab = "Time", ylab = "Value")
```

odin_error_explain *Explain an odin error*

Description

Explain error codes produced by odin. When odin fails to parse your code (e.g., via [odin\(\)](#) or [odin_validate\(\)](#)) it will return an error with a code. You can use `odin_error_explain` to get more information on that code. By default we will print an explanation to the screen, but you can control this behaviour via the `how` argument. All error codes can be found in `vignette("errors")`.

Usage

```
odin_error_explain(code, how = "pretty")
```

Arguments

code	The error code, as a string, in the form Exxxx (a capital "E" followed by four numbers)
how	How to explain the error. Options are pretty (render pretty text in the console), plain (display plain text in the console) and link (browse to the online help).

Value

Nothing, this is called for its side effect only

Examples

```
odin_error_explain("E1006")
```

odin_migrate	<i>Migrate odin code</i>
--------------	--------------------------

Description

Migrate odin code. This function takes a path to existing odin code and writes out migrated code to a new file. It is possible that no code will be migrated, in which case the written contents will be identical to those read.

Usage

```
odin_migrate(path, dest)
```

Arguments

path	Path of the odin code to read
dest	Path of the destination code. It can be the same as path, in which case the file will be overwritten.

Value

Nothing; called for side effects only

Examples

```
# A file 'path' contains odin code using old features:
writeLines(readLines(path))

# Migrate this file in place (by overwriting)
odin_migrate(path, path)

writeLines(readLines(path))
```

odin_package	<i>Update odin code in package</i>
--------------	------------------------------------

Description

Update generated code in a package that uses odin and dust to provide a model. This will generate new dust code in `inst/dust` and from that generate a full model in `src`, and an R interface in `R/dust.R`, along with the `cpp11` attributes that are needed to use the model.

Usage

```
odin_package(path, quiet = FALSE, compatibility = NULL, check_bounds = NULL)
```

Arguments

<code>path</code>	Path to the package root (the directory that contains <code>DESCRIPTION</code>), or any path within that package.
<code>quiet</code>	Logical, indicating if compilation messages from <code>pkgbuild</code> should be displayed. Error messages will be displayed on compilation failure regardless of the value used. If <code>NULL</code> is given, then we take the value from <code>DUST_QUIET</code> if set, or <code>FALSE</code> otherwise.
<code>compatibility</code>	Compatibility mode to use. Valid options are "warning", which updates code that can be fixed, with warnings, and "error", which will error. The option "silent" will silently rewrite code, but this is not recommended for general use as eventually the compatibility mode will be removed (this option is primarily intended for comparing output of <code>odin1</code> and <code>odin2</code> models against old code). The default, <code>NULL</code> , currently corresponds to warning.
<code>check_bounds</code>	Control over static array bounds checking. This is enabled by default, but is prone to false positives, erroring where a read or write appears out of bounds but is actually ok. This argument exists to allow you to disable the check and compile the model anyway. Future versions may allow specific lines to be ignored, which will provide finer control and allow you to use the bits of the checks that are actually helpful. You can also pass <code>TRUE</code> here to mean "error" or <code>FALSE</code> to mean "disabled".

Details

This function is powered by `dust2::dust_package`, and the same pre-requisites apply here:

For your DESCRIPTION file:

- dust2 must be in Imports
- cpp11, dust2 and monty must be in LinkingTo

For your NAMESPACE file:

- you must have a suitable `useDynLib()` call with `.registration = TRUE`

If you do not satisfy these requirements, `dust2::dust_package` will fail with a message indicating actions you should take. Once set up, generally things will keep working.

If you want your packages to build on GitHub actions, or be installable via `remotes::install_github` you should add to your DESCRIPTION:

Remotes: mrc-ide/dust2, mrc-ide/monty

Note that you do not need to include `odin2` itself as a dependency.

Value

Invisibly, the path to the package. However, this function is typically called for its side effect of updating files in `inst/dust` and `src` within this package after you have changed the `odin` code in `inst/odin`.

See Also

`vignette("packaging")` for more details, and `dust2::dust_package()`, which does most of the work here.

Examples

```
# An example package structure
fs::dir_tree(path)

# Generate odin code:
odin_package(path)

# Resulting files:
fs::dir_tree(path)
```

odin_show

*Show generated odin code***Description**

Show generated code from compiling an odin model.

Usage

```
odin_show(
    expr,
    input_type = NULL,
    compatibility = NULL,
    check_bounds = NULL,
    what = NULL
)
```

Arguments

expr	Odin code as the path to a file (a string), a character vector of code, or as an expression (typically within braces {}).
input_type	An optional string describing the type of input for expr - must be one of file, text or expression. If given, this skips the type detection logic and odin will throw an error if the wrong type of input is given. Using this may be beneficial in programmatic environments.
compatibility	Compatibility mode to use. Valid options are "warning", which updates code that can be fixed, with warnings, and "error", which will error. The option "silent" will silently rewrite code, but this is not recommended for general use as eventually the compatibility mode will be removed (this option is primarily intended for comparing output of odin1 and odin2 models against old code). The default, NULL, currently corresponds to warning.
check_bounds	Control over static array bounds checking. This is enabled by default, but is prone to false positives, erroring where a read or write appears out of bounds but is actually ok. This argument exists to allow you to disable the check and compile the model anyway. Future versions may allow specific lines to be ignored, which will provide finer control and allow you to use the bits of the checks that are actually helpful. You can also pass TRUE here to mean "error" or FALSE to mean "disabled".
what	Optional string, being a single method to show. Popular options are update, rhs and compare_data.

Value

A character vector, with class odin_code that has a pretty-print method defined. Returns NULL if what was given but the model lacks this part.

Examples

```
# Show generated code for the whole system
odin_show({
  initial(x) <- 1
  update(x) <- a
  a <- Normal(x, 1)
})

# Just the update method
odin_show({
  initial(x) <- 1
  update(x) <- a
  a <- Normal(x, 1)
}, what = "update")
```

odin_validate

Validate odin code

Description

Validate odin code. This is primarily intended for use within other applications.

Usage

```
odin_validate(
  expr,
  input_type = NULL,
  compatibility = NULL,
  check_bounds = NULL
)
```

Arguments

expr	Odin code as the path to a file (a string), a character vector of code, or as an expression (typically within braces {}).
input_type	An optional string describing the type of input for expr - must be one of file, text or expression. If given, this skips the type detection logic and odin will throw an error if the wrong type of input is given. Using this may be beneficial in programmatic environments.
compatibility	Compatibility mode to use. Valid options are "warning", which updates code that can be fixed, with warnings, and "error", which will error. The option "silent" will silently rewrite code, but this is not recommended for general use as eventually the compatibility mode will be removed (this option is primarily intended for comparing output of odin1 and odin2 models against old code). The default, NULL, currently corresponds to warning.

`check_bounds` Control over static array bounds checking. This is enabled by default, but is prone to false positives, erroring where a read or write appears out of bounds but is actually ok. This argument exists to allow you to disable the check and compile the model anyway. Future versions may allow specific lines to be ignored, which will provide finer control and allow you to use the bits of the checks that are actually helpful. You can also pass `TRUE` here to mean "error" or `FALSE` to mean "disabled".

Value

A list with elements:

- `success`: boolean, `TRUE` if validation was successful
- `result`: Metadata about the model; see Details above for the format.
- `error`: Either `NULL` (if `success` is `TRUE`) or an error; see Details above for interpreting this value.
- `compatibility`: A `data.frame` of compatibility issues. This is formatted similarly to `src` within `error` (see above), but also includes `type` (a code for the compatibility issue), `description` (a human-readable description of the issue), `original` (the original expression) and `value` (the final expression).

The intention is that we won't throw generally from this function.

Result

On successful validation, we return a list with metadata about the model. Currently this contains:

- `time`: The time mode of the model (a string of either "discrete" or "continuous")
- `parameters`: A `data.frame` describing parameters. Currently the only column is `name`.
- `variables`: A `data.frame` describing the model variables. Currently the only column is `name`.
- `data`: A `data.frame` describing data used by the model (if it supports this). Currently the only column is `name`.

Errors

Most errors will have class `odin_parse_error`. These will print context information if rethrown. They have fields:

- `message`: The headline error message
- `code`: The odin error code, as listed in `vignette("errors")`, and used by [odin_error_explain](#)
- `src`: Source information about the error. This is a `data.frame` with columns `index` (the expression number), `expr` (a list column with the expression), `start` (the starting line; possibly `NA`), `end` (the finishing line; possibly `NA`), `str` (the string containing the literal value of the expression; possibly `NA`) and `migrated` (a logical, indicating if the source has been automatically migrated from odin1 code). If any of `start`, `end` or `str` is `NA`, all will be, for all rows.

You can get the full rendered message using `conditionMessage()` on the error object.

Examples

```
# A successful validation:
odin_validate({
  initial(x) <- 1
  deriv(x) <- a
  a <- parameter()
})

# A failure:
odin_validate({
  initial(x) <- 1
  deriv(x) <- a
})

# Migration warnings
odin_validate({
  initial(x)
  deriv(x) <- a
  a <- user()
})
```

Index

`conditionMessage()`, [9](#)

`dust2::dust_package`, [6](#)

`dust2::dust_system_create`, [3](#)

`odin`, [2](#)

`odin()`, [3](#)

`odin_error_explain`, [3](#), [9](#)

`odin_migrate`, [4](#)

`odin_package`, [5](#)

`odin_show`, [7](#)

`odin_validate`, [8](#)

`odin_validate()`, [3](#)

`pkgbuild::compile_dll`, [2](#)

`tempfile()`, [2](#)