

# Package ‘odin’

July 2, 2019

**Title** ODE Generation and Integration

**Version** 1.0.1

**Description** Generate systems of ordinary differential equations (ODE) and integrate them, using a domain specific language (DSL). The DSL uses R's syntax, but compiles to C in order to efficiently solve the system. A solver is not provided, but instead interfaces to the packages 'deSolve' and 'dde' are generated. With these, while solving the differential equations, no allocations are done and the calculations remain entirely in compiled code. Alternatively, a model can be transpiled to R for use in contexts where a C compiler is not present. After compilation, models can be inspected to return information about parameters and outputs, or intermediate values after calculations. 'odin' is not targeted at any particular domain and is suitable for any system that can be expressed primarily as mathematical expressions. Additional support is provided for working with delays (delay differential equations, DDE), using interpolated functions during interpolation, and for integrating quantities that represent arrays.

**License** MIT + file LICENSE

**LazyData** true

**URL** <https://github.com/mrc-ide/odin>

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

**Imports** R6, cinterpolate (>= 1.0.0), crayon, deSolve, digest, jsonlite, ring

**Suggests** dde (>= 1.0.0), jsonvalidate (>= 1.1.0), knitr, rmarkdown, testthat

**VignetteBuilder** knitr

**RoxygenNote** 6.1.1

**Encoding** UTF-8

**Language** en-GB

**NeedsCompilation** no

**Author** Rich FitzJohn [aut, cre],  
 Thibaut Jombart [ctb],  
 Imperial College of Science, Technology and Medicine [cph]

**Maintainer** Rich FitzJohn <rich.fitzjohn@gmail.com>

**Repository** CRAN

**Date/Publication** 2019-07-02 09:00:03 UTC

## R topics documented:

can_compile . . . . .	2
odin . . . . .	3
odin_build . . . . .	5
odin_ir . . . . .	6
odin_ir_deserialise . . . . .	7
odin_options . . . . .	8
odin_package . . . . .	9
odin_parse . . . . .	10
odin_validate . . . . .	11
<b>Index</b>	<b>14</b>

---

can_compile	<i>Test if compilation is possible</i>
-------------	----------------------------------------

---

### Description

Test if compilation appears possible. This is used in some examples, and tries compiling a trivial C program with R CMD SHLIB. Results are cached between runs within a session so this should be fast to rely on.

### Usage

```
can_compile(verbose = FALSE, refresh = FALSE)
```

### Arguments

verbose	Be verbose when running commands?
refresh	Try again to compile, skipping the cached value?

### Details

If this function believes you can't compile, and if gcc can't be found on the path, a diagnostic message will be printed. This will of course not be very interesting if you use a different compiler to gcc! But the most likely people affected here are Windows users; if you get this ensure that you have rtools installed. If you have devtools installed, devtools::find\_rtools() may be helpful for diagnosing compiler issues.

**Value**

A logical scalar

**Examples**

```
can_compile() # will take ~0.1s the first time
can_compile() # should be basically instantaneous
```

---

 odin

*Create an odin model*


---

**Description**

Create an odin model from a file, text string(s) or expression. The `odin_` version is a "standard evaluation" escape hatch.

**Usage**

```
odin(x, verbose = NULL, target = NULL, workdir = NULL,
      validate = NULL, pretty = NULL, skip_cache = NULL,
      compiler_warnings = NULL, no_check_unused_equations = NULL,
      no_check_naked_index = NULL)
```

```
odin_(x, verbose = NULL, target = NULL, workdir = NULL,
       validate = NULL, pretty = NULL, skip_cache = NULL,
       compiler_warnings = NULL, no_check_unused_equations = NULL,
       no_check_naked_index = NULL)
```

**Arguments**

<code>x</code>	Either the name of a file to read, a text string (if length is greater than 1 elements will be joined with newlines) or an expression.
<code>verbose</code>	Logical scalar indicating if the compilation should be verbose. Defaults to the value of the option <code>odin.verbose</code> or <code>FALSE</code> otherwise.
<code>target</code>	Compilation target. Options are "c" and "r", defaulting to the option <code>odin.target</code> or "c" otherwise.
<code>workdir</code>	Directory to use for any generated files. This is only relevant for the "c" target. Defaults to the value of the option <code>odin.workdir</code> or <code>tempdir()</code> otherwise.
<code>validate</code>	Validate the model's intermediate representation against the included schema. Normally this is not needed and is intended primarily for development use. Defaults to the value of the option <code>odin.validate</code> or <code>FALSE</code> otherwise.
<code>pretty</code>	Pretty-print the model's intermediate representation. Normally this is not needed and is intended primarily for development use. Defaults to the value of the option <code>odin.pretty</code> or <code>FALSE</code> otherwise.

<code>skip_cache</code>	Skip odin's cache. This might be useful if the model appears not to compile when you would expect it to. Hopefully this will not be needed often. Defaults to the option <code>odin.skip_cache</code> or <code>FALSE</code> otherwise.
<code>compiler_warnings</code>	Logical scalar indicating if compiler warnings should be converted to R warnings. If this is <code>TRUE</code> , then if any compiler warnings are generated, the compiler output will be displayed (regardless of the value of <code>verbose</code> ) within an R warning (suppressible via <code>suppressWarnings</code> and catchable via <code>tryCatch</code> ). The default is to default to <code>FALSE</code> unless the global option <code>odin.compiler_warnings</code> is set to <code>TRUE</code> (set with <code>options(odin.compiler_warnings = TRUE)</code> ). The default may change to <code>TRUE</code> in future. Warnings are currently a mix of ambiguous syntax in your model (worth fixing) and limitations in the code that odin generates (which you can't fix but I will get on to over time). What is flagged will depend strongly on your platform and what is in your <code>Makevars</code> . I develop odin with <code>-Wall -Wextra -pedantic</code> and still see warnings with both gcc and clang. The compiler output is very simple and may not work on all platforms. Defaults to the option <code>odin.compiler_warnings</code> or <code>FALSE</code> otherwise.
<code>no_check_unused_equations</code>	If <code>TRUE</code> , then don't print messages about unused variables. Defaults to the option <code>odin.no_check_unused_equations</code> or <code>FALSE</code> otherwise.
<code>no_check_naked_index</code>	If <code>TRUE</code> , then if an index variable ( <code>i</code> , <code>j</code> , ...) is used outside of an array subset (e.g., <code>x[] &lt;- i</code> ) then a notice is printed. The behaviour of this functionality changed in odin version <code>0.2.0</code> and this flag is intended to notify users about the change. See <a href="https://github.com/mrc-ide/odin/issues/136">https://github.com/mrc-ide/odin/issues/136</a> for more information. Defaults to the option <code>odin.no_check_naked_index</code> or <code>FALSE</code> otherwise.

## Details

*Do not use `odin::odin` in a package; you almost certainly want to use `odin_package` instead; see the `odin_package` vignette for more information.*

A generated model can return information about itself; `odin_ir`

## Value

A function that can generate the model

## User parameters

If the model accepts user parameters, then the parameter to the constructor or the `set_user` method can be used to control the behaviour when unknown user actions are passed into the model. Possible values are the strings `stop` (throw an error), `warning` (issue a warning but keep going), `message` (print a message and keep going) or `ignore` (do nothing). Defaults to the option `odin.unused_user_action`, or `warning` otherwise. The default behaviour prior to odin version `0.2.0` was equivalent to `ignore`.

## Delay equations with dde

When generating a model one must chose between using the `dde` package to solve the system or the default `deSolve`. Future versions may allow this to switch when using `run`, but for now this requires

tweaking the generated code to a point where one must decide at generation. dde implements only the Dormand-Prince 5th order dense output solver, with a delay equation solver that may perform better than the solvers in deSolve. For non-delay equations, deSolve is very likely to outperform the simple solver implemented.

### Author(s)

Rich FitzJohn

### Examples

```
## Compile the model; exp_decay here is an R6ClassGenerator and will
## generate instances of a model of exponential decay:
exp_decay <- odin::odin({
  deriv(y) <- -0.5 * y
  initial(y) <- 1
}, target = "r")

## Generate an instance; there are no parameters here so all instances
## are the same and this looks a bit pointless. But this step is
## required because in general you don't want to have to compile the
## model every time it is used (so the generator will go in a
## package).
mod <- exp_decay()

## Run the model for a series of times from 0 to 10:
t <- seq(0, 10, length.out = 101)
y <- mod$run(t)
plot(y, xlab = "Time", ylab = "y", main = "", las = 1)
```

---

odin\_build

*Build an odin model generator from its IR*

---

### Description

Build an odin model generator from its intermediate representation, as generated by [odin\\_parse](#). This function is for advanced use.

### Usage

```
odin_build(x, options = NULL)
```

### Arguments

x	An odin ir (json) object or output from <a href="#">odin_validate</a> .
options	Options to pass to the build stage (see <a href="#">odin_options</a> )

## Details

In applications that want to inspect the intermediate representation rather before compiling, rather than directly using `odin`, use either `odin_parse` or `odin_validate` and then pass the result to `odin_build`.

The return value of this function includes information about how long the compilation took, if it was successful, etc, in the same style as `odin_validate`:

**success** Logical, indicating if compilation was successful

**elapsed** Time taken to compile the model, as a `proc_time` object, as returned by `proc.time`.

**output** Any output produced when compiling the model (only present if compiling to C, and if the cache was not hit).

**model** The model itself, as an `odin_generator` object, as returned by `odin`.

**ir** The intermediate representation.

error Any error thrown during compilation

## See Also

`odin_parse`, which creates intermediate representations used by this function.

## Examples

```
##' # Parse a model of exponential decay
ir <- odin::odin_parse({
  deriv(y) <- -0.5 * y
  initial(y) <- 1
})

# Compile the model:
options <- odin::odin_options(target = "r")
res <- odin::odin_build(ir, options)

# All results:
res

# The model:
mod <- res$model()
mod$run(0:10)
```

---

odin\_ir

*Return detailed information about an odin model*

---

## Description

Return detailed information about an odin model. This is the mechanism through which `coef` works with `odin`.

**Usage**

```
odin_ir(x, parsed = FALSE)
```

**Arguments**

x	An <code>odin_generator</code> function, as created by <a href="#">odin</a>
parsed	Logical, indicating if the representation should be parsed and converted into an R object. If FALSE we return a json string.

**Warning**

The returned data is subject to change for a few versions while I work out how we'll use it.

**Examples**

```
exp_decay <- odin::odin({
  deriv(y) <- -0.5 * y
  initial(y) <- 1
}, target = "r")
odin::odin_ir(exp_decay)
coef(exp_decay)
```

---

odin\_ir\_deserialise    *Deserialise odin's IR*

---

**Description**

Deserialise odin's intermediate model representation from a json string into an R object. Unlike the json, there is no schema for this representation. This function provides access to the same deserialisation that odin uses internally so may be useful in applications.

**Usage**

```
odin_ir_deserialise(x)
```

**Arguments**

x	An intermediate representation as a json string
---	-------------------------------------------------

**Value**

A named list

**See Also**

[odin\\_parse](#)

## Examples

```
# Parse a model of exponential decay
ir <- odin::odin_parse({
  deriv(y) <- -0.5 * y
  initial(y) <- 1
})
# Convert the representation to an R object
odin::odin_ir_deserialise(ir)
```

---

odin\_options

*Odin options*

---

## Description

For lower-level odin functions [odin\\_parse](#), [odin\\_validate](#) we accept a list of options rather than individually named options.

## Usage

```
odin_options(verbose = NULL, target = NULL, workdir = NULL,
  validate = NULL, pretty = NULL, skip_cache = NULL,
  compiler_warnings = NULL, no_check_unused_equations = NULL,
  no_check_naked_index = NULL, options = NULL)
```

## Arguments

verbose	Logical scalar indicating if the compilation should be verbose. Defaults to the value of the option <code>odin.verbose</code> or FALSE otherwise.
target	Compilation target. Options are "c" and "r", defaulting to the option <code>odin.target</code> or "c" otherwise.
workdir	Directory to use for any generated files. This is only relevant for the "c" target. Defaults to the value of the option <code>odin.workdir</code> or <code>tempdir()</code> otherwise.
validate	Validate the model's intermediate representation against the included schema. Normally this is not needed and is intended primarily for development use. Defaults to the value of the option <code>odin.validate</code> or FALSE otherwise.
pretty	Pretty-print the model's intermediate representation. Normally this is not needed and is intended primarily for development use. Defaults to the value of the option <code>odin.pretty</code> or FALSE otherwise.
skip_cache	Skip odin's cache. This might be useful if the model appears not to compile when you would expect it to. Hopefully this will not be needed often. Defaults to the option <code>odin.skip_cache</code> or FALSE otherwise.
compiler_warnings	Logical scalar indicating if compiler warnings should be converted to R warnings. If this is TRUE, then if any compiler warnings are generated, the compiler



output will be displayed (regardless of the value of `verbose`) within an R warning (suppressible via `suppressWarnings` and catchable via `tryCatch`). The default is to default to `FALSE` unless the global option `odin.compiler_warnings` is set to `TRUE` (set with `options(odin.compiler_warnings = TRUE)`). The default may change to `TRUE` in future. Warnings are currently a mix of ambiguous syntax in your model (worth fixing) and limitations in the code that odin generates (which you can't fix but I will get on to over time). What is flagged will depend strongly on your platform and what is in your `Makevars`. I develop odin with `-Wall -Wextra -pedantic` and still see warnings with both `gcc` and `clang`. The compiler output is very simple and may not work on all platforms. Defaults to the option `odin.compiler_warnings` or `FALSE` otherwise.

`no_check_unused_equations`

If `TRUE`, then don't print messages about unused variables. Defaults to the option `odin.no_check_unused_equations` or `FALSE` otherwise.

`no_check_naked_index`

If `TRUE`, then if an index variable (`i`, `j`, ...) is used outside of an array subset (e.g., `x[] <- i`) then a notice is printed. The behaviour of this functionality changed in odin version `0.2.0` and this flag is intended to notify users about the change. See <https://github.com/mrc-ide/odin/issues/136> for more information. Defaults to the option `odin.no_check_naked_index` or `FALSE` otherwise.

`options`

Named list of options. If provided, then all other options are ignored.

## Examples

```
odin_options()
```

---

<code>odin_package</code>	<i>Create odin model in a package</i>
---------------------------	---------------------------------------

---

## Description

Create an odin model within an existing package.

## Usage

```
odin_package(path_package)
```

## Arguments

`path_package` Path to the package root (the directory that contains `DESCRIPTION`)

## Details

I am resisting the urge to actually create the package here. There are better options than I can come up with; for example `devtools::create`, `pkgkitten::kitten`, `mason::mason`, or creating `DESCRIPTION` files using `desc`. What is required here is that your package:

- Lists odin in Imports:
- Includes useDynLib{<your package name>} in NAMESPACE (possibly via a roxygen comment @useDynLib <your package name>)
- To avoid a NOTE in R CMD check, import something from odin in your namespace (e.g., importFrom("odin", "odin") or roxygen @importFrom(odin, odin)

Point this function at the package root (the directory containing DESCRIPTION and it will write out files src/odin.c and odin.R. These files will be overwritten without warning by running this again.

There are a few unresolved issues with this approach, notably activating "native symbol registration", and the interaction with packages such as Rcpp that automatically collate a list of symbols. The mechanism may change in a future version, though the interface (with source files in inst/odin will remain the same.

### Examples

```
path <- tempfile()
dir.create(path)

src <- system.file("examples/package", package = "odin", mustWork = TRUE)
file.copy(src, path, recursive = TRUE)
pkg <- file.path(path, "package")

# The package is minimal:
dir(pkg)

# But contains odin files in inst/odin
dir(file.path(pkg, "inst/odin"))

# Compile the odin code in the package
odin::odin_package(pkg)

# Which creates the rest of the package structure
dir(pkg)
dir(file.path(pkg, "R"))
dir(file.path(pkg, "src"))
```

---

odin\_parse

*Parse an odin model*

---

### Description

Parse an odin model, returning an intermediate representation. The `odin_parse_` version is a "standard evaluation" escape hatch.

### Usage

```
odin_parse(x, type = NULL, options = NULL)
```

```
odin_parse_(x, options = NULL, type = NULL)
```

## Arguments

x	An expression, character vector or filename with the odin code
type	An optional string indicating the the type of input - must be one of expression, file or text if provided. This skips the type detection code used by odin and makes validating user input easier.
options	odin options; see <a href="#">odin_options</a> . The primary options that affect the parse stage are validate and pretty.

## Details

A schema for the intermediate representation is available in the package as `schema.json`. It is subject to change at this point.

## See Also

[odin\\_validate](#), which wraps this function where parsing might fail, and [odin\\_build](#) for building odin models from an intermediate representation.

## Examples

```
# Parse a model of exponential decay
ir <- odin::odin_parse({
  deriv(y) <- -0.5 * y
  initial(y) <- 1
})

# This is odin's intermediate representation of the model
ir

# If parsing odin models programmatically, it is better to use
# odin_parse_; construct the model as a string, from a file, or as a
# quoted expression:
code <- quote({
  deriv(y) <- -0.5 * y
  initial(y) <- 1
})

odin::odin_parse_(code)
```

---

odin\_validate

*Validate an odin model*

---

## Description

Validate an odin model. This function is closer to [odin\\_parse\\_](#) than [odin\\_parse](#) because it does not do any quoting of the code. It is primarily intended for use within other applications.

**Usage**

```
odin_validate(x, type = NULL, options = NULL)
```

**Arguments**

x	An expression, character vector or filename with the odin code
type	An optional string indicating the the type of input - must be one of expression, file or text if provided. This skips the type detection code used by odin and makes validating user input easier.
options	odin options; see <a href="#">odin_options</a> . The primary options that affect the parse stage are validate and pretty.

**Details**

odin\_validate will always return a list with the same elements:

**success** A boolean, TRUE if validation was successful

**result** The intermediate representation, as returned by [odin\\_parse\\_](#), if the validation was successful, otherwise NULL

**error** An error object if the validation was unsuccessful, otherwise NULL. This may be a classed odin error, in which case it will contain source location information - see the examples for details.

**messages** A list of messages, if the validation returned any. At present this is only non-fatal information about unused variables.

**Author(s)**

Rich FitzJohn

**Examples**

```
# A successful validation:
odin::odin_validate(c("deriv(x) <- 1", "initial(x) <- 1"))

# A complete failure:
odin::odin_validate("")

# A more interesting failure
code <- c("deriv(x) <- a", "initial(x) <- 1")
res <- odin::odin_validate(code)
res

# The object 'res$error' is an 'odin_error' object:
res$error

# It contains information that might be used to display to a
# user information about the error:
unclass(res$error)
```

```
# Notes are raised in a similar way:  
code <- c("deriv(x) <- 1", "initial(x) <- 1", "a <- 1")  
res <- odin::odin_validate(code)  
res$messages[[1]]
```

# Index

can\_compile, [2](#)  
coef, [6](#)

odin, [3](#), [6](#), [7](#)  
odin\_(odin), [3](#)  
odin\_build, [5](#), [11](#)  
odin\_ir, [4](#), [6](#)  
odin\_ir\_deserialise, [7](#)  
odin\_options, [5](#), [8](#), [11](#), [12](#)  
odin\_package, [4](#), [9](#)  
odin\_parse, [5-8](#), [10](#), [11](#)  
odin\_parse\_, [11](#), [12](#)  
odin\_parse\_(odin\_parse), [10](#)  
odin\_validate, [5](#), [6](#), [8](#), [11](#), [11](#)

proc.time, [6](#)