# Package: offartmb (via r-universe)

September 15, 2024

**Title** Use offarray code with RTMB

**Version** 1.0.25

**Description** Helper functions to allow offarray code to work directly
under RTMB

**Author** Mark Bravington <`markb2@summerinsouth.net`>

**Maintainer** Mark Bravington <`markb2@summerinsouth.net`>

**Imports** utils, mvbutils, offarray (>= 2.0), RTMB

**License** MIT

**Repository** https://markbravington.r-universe.dev

**RemoteUrl** https://github.com/markbravington/offartmb

**RemoteRef** HEAD

**RemoteSha** 0561e17f8c5bfeacae241dc6f89d58e5e9881b82

## Contents

Obinary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

**Index** **4**

---

`Obinary` *Overloading glue between offarray and other packages such as RTMB*

---

### Description

Package **offartmb** lets you use `offarray` code directly in package **RTMB**. All you have to do is:

- make sure you have done `library( offartmb)` *after* `library( offarray)`
- make sure the body of your objective function, and any hand-written sub-functions that is calls, is wrapped in a call to `reclasso` which is in package **offarray**. As its documentation says, there's no downside to doing that.

- make sure you use `REPORTO( thing_I_want_to_keep)` to preserve interesting variables computed inside your function (like REPORT() in TMB or indeed package **RTMB**).

Then your function `myfun` will still run fine outside `RTMB`, but you should also be able to run `myfun_rtmb <- RTMB::MakeADFun( myfun, <paramvals>)` and everything will be copacetic thereafter.

## Usage

```
define_repops( ...)
# You would never just Obinary on its own like this...
Obinary(op, e1, e2, ..., allow_unary = FALSE)
```

## Arguments

| | |
|---|---|
| `...` | in `define_repops`, a named list of replacement operator/functions and expressions to replace them with. The expressions normally need to be wrapped in `quote`— you don't want `define_repops` itself to evaluate them. See `offartmb:::.onLoad` for example. If empty, the current repops will be returned. In `Obinary`, `...` can contain additional arguments for op, which will be passed to it unchanged. This is not meaningful for pure operators such as "*", but conceivably useful for eg statistical distributions where the mean, or mean-and-variance, might be S4; often the user might want to pass an extra argument such as "log=FALSE" or "df=5". |
| `op` | Name of an "operator". Usually eg "+" or similar but potentially any function which should dispatch based on its first two arguments. |
| `e1, e2` | Arguments whose class to dispatch on |
| `allow_unary` | A few functions, such as "-", have a legitimate unary variant; `-x` makes sense on its own, whereas `*x` doesn't. |

## Details

You probably do not want to be reading this.

But, well, here you still are, so here it is. As you know, S3 classes (such as `offarray`) don't always play nicely with S4 classes (such as `advector` in package **RTMB**); the latter can be big bullies in terms of insisting that *Their* class comes First, leading to downstream woe. So if you want S3 code to run both on "normal" R objects and on objects that might be S4, there is work to do— either by you, or ideally behind-the-scenes automatically, which is where package **offartmb** can help. Particular problems occur with "double dispatch" on operators such as addition, where R's built-in S3 dispatch rules are well-known to be borked. One option is to S4-ify the S3 class, and deal specifically with multiple inheritance, but that's a lot of work. Another option is to use something like `offarray::reclasso` (qv) to modify the code that is being run, to replace the base-R calls to eg "+" with calls to functions that know how to dispatch properly.

`reclasso` is an S3 generic dispatching on its by argument, and the default version actually makes no modifications. But there is a method `offartmb:::reclasso_advector` which (should) work on `advector`-class objects from package **RTMB**, ie when `RTMB::MakeADFun` (qv) is running your code. `reclasso_advector` makes some additional tweaks as well, eg to `REPORTO` for stashing results. If you wanted `offarray` to work with some other non-'RTMB' S4 package, you would need to write a similar generic. The additional tweaks are likely package-specific, but for operator-replacement the versions in `offartmb` itself might be usable as-is; read on.

It is pretty unlikely that you will *ever* need to tinker with any of this yourself, but I need to document at least one function in order for this package to install smoothly! Anyway, even with `RTMB`, you *might* conceivably need to add your own replacement operator/function for some weird thing that doesn't work out-of-the-box with `offarray` (although a lot of functions in `RTMB` actually seem to work OK). You can do so in two steps with `define_repops`, eg via

```
  .besselZfun <- function( e1, e2, ...) Obinary( 'besselZ', e1, e2, ...)
  define_repops( besselZ=quote( .besselZfun))
```

The basic trick for most operators is to remove the `offarray` class from the operand(s), then call the base-R operator which will normally lead to some S4 method being invoked, then add back the `offarray` class and its dimensions etc to the result. Since many operators/functions follow a similar pattern, the function `Obinary` can be used to easily generate replacement operators. For example, the replacement for "*") is in effect

```
  offartmb:::.Otimes <- function( e1, e2) Obinary( "*", e1, e2, FALSE)
```

`Obinary` is not actually specific to RTMB stuff, and might be useful in the event anyone ever needs to add similar functionality between `offarray` and some S4 package that is not `RTMB`.

## Value

`Obinary` returns a new function. `define.repops` normally returns (invisibly) the original set of replacement operator/functions.

## See Also

`offarray::reclasso`, `mvbutils::REPORTO`, `RTMB::MakeADFun`

## Examples

```
## should have one, I guess...
```

# Index

∗ **misc**
    Obinary, 1

define_repops (Obinary), 1

Obinary, 1
offartmb (Obinary), 1

reclasso, *1*