Package: RDO (via r-universe)

October 13, 2024

Title Reproducible Data Objects

Version 0.1.0

Description A Reproducible Data Object (RDO) encapsulates both data and R code needed to reproduce those data. Each RDO can have other RDOs as dependencies. RDOs can be composed into complex tree hierarchies. Interacting with an RDO tree is similar to interacting with a single RDO. You can (re)run the code and refresh the cache, check status, validate if the code still gives the same cached data, clear data cache, access code and data cache of any of the dependencies. RDOs can be cloned and code of cloned dependencies can be modified.

License MIT + file LICENSE

URL https://github.com/openpharma/RDO#readme

BugReports https://github.com/openpharma/RDO/issues

Encoding UTF-8

LazyData true

Imports purrr, data.table

Suggests visNetwork, testthat, knitr, rmarkdown

RoxygenNote 7.0.2

VignetteBuilder knitr

Repository https://pharmaverse.r-universe.dev

RemoteUrl https://github.com/openpharma/RDO

RemoteRef HEAD

RemoteSha 19aed10f427371c30db3f890a4ee127923c1134e

Contents

RDO *RDO (Reproducible Data Object)*

Description

Create and interact with Reproducible Data Objects.

Methods

Public methods:

- RDO$print()
- RDO$new()
- RDO$get_status()
- RDO$get_name()
- RDO$has_dependencies()
- RDO$add_dependencies()
- RDO$get_dependencies()
- RDO$get_dependency_register()
- RDO$plot_dependencies()
- RDO$get_code()
- RDO$print_code()
- RDO$run()
- RDO$is_validated()
- RDO$invalidate()
- RDO$validate()
- RDO$lock()
- RDO$unlock()
- RDO$is_locked()
- RDO$prune_cache()
- RDO$prune_dependencies()
- RDO$clone()

Method `print()`**:** Default print method of current RDO status

Usage:

```
RDO$print(...)
```

Arguments:

`...` Other params for print function.

Method `new()`**:** Creating a new RDO object,

Usage:

```
RDO$new(name, dependencies = list())
```

Arguments:

`name` Unique name of the object.

`dependencies` An RDO object or a list of RDO objects.

Returns: A new 'RDO' object.

Method `get_status()`**:** Getting current status.

Usage:

```
RDO$get_status()
```

Returns: A list.

Method `get_name()`**:** Getting object name.

Usage:

```
RDO$get_name()
```

Returns: A character with object name.

Method `has_dependencies()`**:** Checking if object has dependencies.

Usage:

```
RDO$has_dependencies()
```

Returns: TRUE if has or FALSE if not.

Method `add_dependencies()`**:** Adding new or update existing RDO dependencies.

Usage:

```
RDO$add_dependencies(dependencies = list())
```

Arguments:

`dependencies` An RDO object or a list of RDO objects.

Returns: The RDO object (self) returned invisibly.

Method `get_dependencies()`**:** Getting dependencies of the object.

Usage:

```
RDO$get_dependencies(deep = FALSE)
```

Arguments:

`deep` A logical. Should the function return only direct dependencies (FALSE) or also deep indirect dependencies (dependencies of dependencies). Default is FALSE.

Returns: A named list of RDO dependencies with unique names.

Method `get_dependency_register()`**:** Getting dependency register showing which RDO is a direct parent for other RDO dependencies.

Usage:

```
RDO$get_dependency_register()
```

Returns: A data.frame.

Method `plot_dependencies()`**:** Plotting the tree of RDO dependencies. Needs `visNetwork` package for plotting.

Usage:

```
RDO$plot_dependencies()
```

Method `get_code()`**:** Getting reproducible R code.

Usage:

```
RDO$get_code(deep = FALSE)
```

Arguments:

`deep` A logical. Should the function return only code for this particular RDO (FALSE) or also from all dependencies. Default is FALSE.

Returns: An R named expression.

Method `print_code()`**:** Printing reproducible R code.

Usage:

```
RDO$print_code(deep = FALSE, verbose = Sys.getenv("RDO_VERBOSE"))
```

Arguments:

`deep` A logical. Should the function return only code for this particular RDO (FALSE) or also from all dependencies. Default is FALSE.

`verbose` A logical. Should the messages be sent to console. If the param is not set, it is read from an environmental variable `RDO_VERBOSE`. If the variable is not set, than the default is TRUE.

Returns: A character. Reproducible R code returned invisibly.

Method `run()`**:** Running reproducible R code.

Usage:

```
RDO$run(deep = FALSE, cache = TRUE, verbose = Sys.getenv("RDO_VERBOSE"))
```

Arguments:

`deep` A logical. Should the function run only code for this particular RDO (FALSE) or should it run also dependencies' code if they are not validated. The function is lazy and it checks if all deep dependencies are validated first. If so, there is no need to rerun their code again. Default is FALSE.

`cache` A logical. Should the result of code evaluation be cached inside an RDO object. Default is TRUE.

`verbose` A logical. Should the messages be sent to console. If the param is not set, it is read from an environmental variable `RDO_VERBOSE`. If the variable is not set, than the default is TRUE.

Returns: The RDO object (self) returned invisibly.

Method `is_validated()`**:** Checking if RDO is validated. A RDO is validated when the result of running reproducible R code saved inside the RDO is the same as data cached inside the RDO.

Usage:

```
RDO$is_validated(deep = FALSE, verbose = Sys.getenv("RDO_VERBOSE"))
```

Arguments:

`deep` A logical. Should the function validate only this particular RDO (FALSE) or should it validate also all deep dependencies (TRUE). Default is FALSE.

verbose A logical. Should the messages be sent to console. If the param is not set, it is read from an environmental variable RDO_VERBOSE. If the variable is not set, than the default is TRUE.

Returns: A logical. TRUE if RDO is validated, FALSE if not.

Method `invalidate()`**:** Invalidating the RDO explicitly by setting the 'is_validated' status to 'FALSE'.

Usage:

```
RDO$invalidate(deep = FALSE, verbose = Sys.getenv("RDO_VERBOSE"))
```

Arguments:

deep A logical. Should the function invalidate only this particular RDO (FALSE) or should it invalidate also all deep dependencies (TRUE). Default is FALSE.

verbose A logical. Should the messages be sent to console. If the param is not set, it is read from an environmental variable RDO_VERBOSE. If the variable is not set, than the default is TRUE.

Returns: The RDO object (self) returned invisibly.

Method `validate()`**:** Validating the RDO explicitly by running deep reproducible R code and checking if the result is the same as cached data inside the RDO.

Usage:

```
RDO$validate(deep = FALSE, verbose = Sys.getenv("RDO_VERBOSE"))
```

Arguments:

deep A logical. Should the function validate only this particular RDO (FALSE) or should it validate also all deep dependencies (TRUE). Default is FALSE.

verbose A logical. Should the messages be sent to console. If the param is not set, it is read from an environmental variable RDO_VERBOSE. If the variable is not set, than the default is TRUE.

Returns: The RDO object (self) returned invisibly.

Method `lock()`**:** Locking the RDO object. When RDO object is locked you cannot change the R code and data cache saved inside the object.

Usage:

```
RDO$lock(deep = FALSE, verbose = Sys.getenv("RDO_VERBOSE"))
```

Arguments:

deep A logical. Should the function lock only this particular RDO (FALSE) or should it lock also all deep dependencies (TRUE). Default is FALSE.

verbose A logical. Should the messages be sent to console. If the param is not set, it is read from an environmental variable RDO_VERBOSE. If the variable is not set, than the default is TRUE.

Returns: The RDO object (self) returned invisibly.

Method `unlock()`**:** Unlocking previously locked RDO object.

Usage:

```
RDO$unlock(deep = FALSE, verbose = Sys.getenv("RDO_VERBOSE"))
```

Arguments:

`deep` A logical. Should the function unlock only this particular RDO (FALSE) or should it unlock also all deep dependencies (TRUE). Default is FALSE.

`verbose` A logical. Should the messages be sent to console. If the param is not set, it is read from an environmental variable `RDO_VERBOSE`. If the variable is not set, than the default is TRUE.

Returns: The RDO object (self) returned invisibly.

Method `is_locked()`**:** Checking if an RDO object is locked.

Usage:

```
RDO$is_locked(deep = FALSE, verbose = Sys.getenv("RDO_VERBOSE"))
```

Arguments:

`deep` A logical. Should the function check only this particular RDO (FALSE) or should it check also all deep dependencies (TRUE). Default is FALSE.

`verbose` A logical. Should the messages be sent to console. If the param is not set, it is read from an environmental variable `RDO_VERBOSE`. If the variable is not set, than the default is TRUE.

Returns: A logical. TRUE if RDO is locked, FALSE if not.

Method `prune_cache()`**:** Pruning (clearing) RDO data cache by setting cache to NULL. It can save memory when we no longer need to keep cache in dependencies.

Usage:

```
RDO$prune_cache(deep = FALSE, verbose = Sys.getenv("RDO_VERBOSE"))
```

Arguments:

`deep` A logical. Should the function prune cache of only this particular RDO (FALSE) or should it prune cache also of deep dependencies (TRUE). If an RDO is locked the cache in this particular RDO is not pruned. Default is FALSE.

`verbose` A logical. Should the messages be sent to console. If the param is not set, it is read from an environmental variable `RDO_VERBOSE`. If the variable is not set, than the default is TRUE.

Returns: The RDO object (self) returned invisibly.

Method `prune_dependencies()`**:** Pruning RDO dependencies by ensuring that RDO objects in deep dependencies with the same name point to the same RDO objects. This type of pruning may be useful after deep clonning of complex RDO tree with duplicated dependencies.

Usage:

```
RDO$prune_dependencies(verbose = Sys.getenv("RDO_VERBOSE"))
```

Arguments:

`verbose` A logical. Should the messages be sent to console. If the param is not set, it is read from an environmental variable `RDO_VERBOSE`. If the variable is not set, than the default is TRUE.

Returns: The RDO object (self) returned invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Index