Package 'lrequire'

August 29, 2016

Type Package

Title Sources an R ``Module" with Caching & Encapsulation, Returning Exported Vars

Version 0.1.3

Date 2016-02-20

Author Rick Wargo <lrequire@rickwargo.com>

Maintainer Rick Wargo <lrequire@rickwargo.com>

Depends R (>= 3.0.1)

Suggests testthat

Description In the fashion of 'node.js' <https://nodejs.org/>, requires a file, sourcing into the current environment only the variables explicitly specified in the module.exports or exports list variable. If the file was already sourced, the result of the earlier sourcing is returned to the caller.

License MIT + file LICENSE

URL https://github.com/rickwargo/lrequire

BugReports https://github.com/rickwargo/lrequire/issues

LazyData TRUE

RoxygenNote 5.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2016-02-22 08:04:15

R topics documented:

`append.module.paths` *Append/Insert a path into module.paths, similar to append()*

Description

Beware, little error checking is done to see if the indexes are valid. Note the item indexes are 1-based.

Usage

```
append.module.paths(value, after = -1)
```

Arguments

`value`	(relative) path to insert into `module.paths`
`after`	location in `module.paths` to append, 0 for the beginning, defaults to -1 which appends to the end

Value

Nothing is returned.

Examples

```
# Inserts `../R/lib' as the second path to search for modules
append.module.paths('../R/lib', after = 1)
```

`find.first.R` *Returns the path of the first found instance of* `module` *in* `module.path`.

Description

A symbol maybe passed instead of a string for readability. If an expression is passed, it must return a string value.

Usage

```
find.first.R(module, character.only = FALSE, warn.not.found = TRUE)
```

Arguments

module	a string (or symbol) specifying the module to search for existance and readability in the current directory, and if it cannot be found, searches for it in the list of directories specified by module.paths and then through the set of paths with the module using a .R extension, if it was not originally specified.
character.only	a logical value, defaulted to FALSE, that permits an unquoted name to be lrequired. Set this to TRUE when passing a variable to lrequire, requiring a quoted string.
warn.not.found	a logical value, defaulted to TRUE, can be set to not display warning messages when module is not found.

Value

A string consisting of the path the module was first found searching through module.paths.

Examples

```
hide.not.found.warnings()  # don't warn on files not foudn by find.first.R()

# Returns the path to the first found module according to module.paths
hello_ex.path <- find.first.R(hello_ex)
```

---

get.module.cache *Returns the current file cache*

Description

Returns the current file cache

Usage

```
get.module.cache()
```

Value

environment containing the file cache.

Examples

```
cache <- get.module.cache()
```

get.module.paths *Get existing collection of search paths of where to look for modules.*

Description

Get existing collection of search paths of where to look for modules.

Usage

```
get.module.paths()
```

Value

Vector of paths (strings) that specify folders where to search for module files, in order.

Examples

```
# Returns a copy of the current module.paths
paths <- get.module.paths()
```

hide.not.found.warnings *Globally hide warnings when modules are not found*

Description

This is only to be called when handling results manually. This will be overridden by a `warn.not.found` parameter explicitly set by either `lrequire`, `find.first.R`.

Usage

```
hide.not.found.warnings()
```

Value

nothing is returned

Examples

```
# Ensure warnings are not displayed when lrequire cannot find the module
hide.not.found.warnings()
```

`lrequire` *Sources an R module with optional caching for subsequent attempts, exporting specified values*

Description

`lrequire` looks in the current path, and then through a list of predefined paths to search for the given module to source into the current environment, but only making visible specific variables that are "exported" as a list, in a fashion similar to node.js. The caching behaviour can be either suspended or it can re-source files that have changed since the last time the module was cached.

Usage

```
lrequire(module, force.reload = FALSE, character.only = FALSE,
  warn.not.found = TRUE)
```

Arguments

`module`	a string (or expression) that specifies a module to load, with or without an optional .R extension. If the module does not exist in the current directory, it searches for the module in directories listed in `module.paths`, first seaching all directories for the named module, then the module with a .R extension. • ./R_modules/ • ./lib/ • ../R_modules/ • ../lib/ • ~/.R_modules All variables exposed in the module will be hidden in the calling environment, except for what is exposed through module.exports or the exports list variable.
`force.reload`	a logical value, defaulted to FALSE, that can be set to TRUE to disable caching behavior for the module. If the module has already been loaded and cached, setting `force.reload` to TRUE will re-source the module. Setting it again to FALSE will re-source the module if the previous state was TRUE.
`character.only`	a logical value, defaulted to FALSE, that permits an unquoted name to be `lrequire`-d. Set this to TRUE when passing a variable to `lrequire`, requiring a quoted string.
`warn.not.found`	a logical value, defaulted to TRUE, can be set to not display warning messages when module is not found.

Details

`lrequire` operates in a similar principle to modules in node.js - keeping any variables created in the source module isolated from the calling environment, while exposing a select set of values/parameters. The specific values are exposed by setting a named list element in the `exports` variable to the desired value or by assigning `module.exports` a value.

Note this list exposed in `module.exports` should have named items so they can easily be accessed in the calling environment, however that is not necessary if only a single value is being returned.

If values are assigned to both `module.exports` and `exports`, only the values in `module.exports` will be exposed to the caller.

Caching a long-running operation, such as static data retrieval from a database is a good use of the caching capability of `lrequire` during development when the same module is sourced multiple times.

During development, files can be reloaded, even if being cached, if they have been modified after the time they were cached. To enable this behaviour, set the variable `module.change_code` to 1.

To quickly clear lrequire's package environment, unload the package. In RStudio, this can be done by unchecking `lrequire` on the Packages tab. You can also execute the following at the R prompt: `detach("package:lrequire", unload=TRUE)` The next call to `library(lrequire)` will ensure it starts off with a clean slate.

Value

Any values that exist in `module.exports` or, if that does not exist, then the *list* `exports`.

If no module is found, `NA` is returned.

Author(s)

Rick Wargo, <lrequire@rickwargo.com>

Examples

```
hide.not.found.warnings()  # don't warn on files not found by lrequire()

# If the module name is in a character vector, use:
my.module <- 'myplot'
mm <- lrequire(my.module, character.only = TRUE)

say.hello.to <- lrequire(hello_ex)
# say.hello.to('Rick')  # use the say.hello.to() function that was returned by lrequire()
```

---

remove.from.module.cache

Removes module from cache, applying same logic as `find.first.R` to find and remove it

---

Description

Removes module from cache, applying same logic as `find.first.R` to find and remove it

Usage

```
remove.from.module.cache(module, character.only = FALSE)
```

Arguments

module	name of a module, same as the one used in the lrequire method, that will be removed from the cache, such that the next time the module is lrequire'd, it will be read and executed.
character.only	a logical value, defaulted to FALSE, that permits an unquoted name to be lrequired. Set this to TRUE when passing a variable to lrequire, requiring a quoted string.

Value

boolean value yielding success of removal from the cache

Examples

```
remove.from.module.cache(variables)
```

---

remove.module.paths *Remove one or more paths from* module.paths

---

Description

Beware, little error checking is done to see if the indexes are valid. Note the item indexes are 1-based.

Usage

```
remove.module.paths(index, ...)
```

Arguments

index	index of item to be removed from path
...	optional indexes of items to be removed from path

Value

Nothing is returned.

Examples

```
# Removes the2nd and 4th items from module.paths
remove.module.paths(2, 4)
```

reset.module.cache *Resets the module cache, ensuring files are loaded on next require*

Description

Note the chace contains other hidden variables, kept in the cache (or environment). These are not removed, and removing them will conflict with the ability of `lrequire` to perform properly.

Usage

```
reset.module.cache()
```

Value

Nothing is returned

Examples

```
reset.module.cache()
```

show.module.cache *Prints the current file cache*

Description

Prints the current file cache

Usage

```
show.module.cache(all.names = FALSE)
```

Arguments

`all.names`	a logical value. If TRUE, all object names are returned. If FALSE, names which begin with a . are omitted.

Value

Nothing is returned, however, the contents of the module cache are printed to the standard output.

Examples

```
show.module.cache()
show.module.cache(all.names = TRUE)
```

show.not.found.warnings *Globally show warnings when modules are not found*

Description

This is the default behaviour.

Usage

```
show.not.found.warnings()
```

Details

This is only to be called when handling results manually. This will be overridden by a `warn.not.found` parameter explicitly set by either `lrequire` or `find.first.R`.

Value

nothing is returned

Examples

```
# Ensure warnings are displayed when lrequire cannot find the module
show.not.found.warnings()
```

Index