

Package: mcparallelDo (via r-universe)

September 24, 2026

Type Package

Title A Simplified Interface for Running Commands on Parallel Processes

Version 1.0.1

Date 2015-12-07

Author Russell S. Pierce

Maintainer Russell S. Pierce <russell.s.pierce@gmail.com>

Description Provides a function that wraps `mcparallel()` and `mccollect()` from 'parallel' with temporary variables and a task handler. Wrapped in this way the results of an `mcparallel()` call can be returned to the R session when the fork is complete without explicitly issuing a specific `mccollect()` to retrieve the value. Outside of top-level tasks, multiple `mcparallel()` jobs can be retrieved with a single call to `mcparallelDoCheck()`.

License GPL-2

Suggests testthat

RoxygenNote 5.0.1

Imports parallel, R.utils, checkmate (>= 1.6.3), R6

Repository <https://russellpierce.r-universe.dev>

Date/Publication 2016-07-31 20:04:26 UTC

RemoteUrl <https://github.com/russellpierce/mcparalleldo>

RemoteRef HEAD

RemoteSha 2ab16ff03cacb942104aec7c87f6a290521d779a

Contents

<code>mcparallelDo-package</code>	2
<code>checkIfJobStillRunning</code>	2
<code>jobCompleteSelfDestructingHandler</code>	3
<code>mcparallelDo</code>	3

mcparallelDoCheck	4
mcparallelDoManagerClass	5
Index	6

mcparallelDo-package	<i>mcparallelDo-package placeholder</i>
----------------------	---

Description

A repository for a variety of useful functions.

Details

The primary function of this package is mcparallelDo(). To use mcparallelDo(), simply invoke the function with a curly braced wrapped code and the character element name to which you want to assign the results.

checkIfJobStillRunning	<i>checkIfJobStillRunning</i>
------------------------	-------------------------------

Description

checkIfJobStillRunning

Usage

checkIfJobStillRunning(targetJob, targetValue, verbose, targetEnvironment)

Arguments

- targetJob (character) The job name
- targetValue (character) The return variable name
- verbose (logical) Whether a message will be generated when complete
- targetEnvironment (environment) Target environment

Value

logical; TRUE if still running; FALSE if not running

```
jobCompleteSelfDestructingHandler
      jobCompleteDestructingHandler
```

Description

Creates a callback handler function that can be added via `addTaskCallback()`. These functions run at the end of each completed R statement. This particular handler watches for the completion of the target job, which is created via `mcpParallel()`

Usage

```
jobCompleteSelfDestructingHandler(targetJob, targetValue, verbose,
      targetEnvironment)
```

Arguments

<code>targetJob</code>	(character) Name of the <code>mcpParallel</code> job variable that is waiting for a result
<code>targetValue</code>	A character element indicating the variable that the result of that job should be assigned to <code>targetEnvironment</code>
<code>verbose</code>	A boolean element; if TRUE the completion of the fork expr will be accompanied by a message
<code>targetEnvironment</code>	The environment in which you want <code>targetValue</code> to be created

Value

callback handler function

```
mcpParallelDo      mcpParallelDo
```

Description

This function creates a fork, sets the variable named `targetValue` in the `targetEnvironment` to NULL, evaluates a segment of code evaluated in the fork, and the result of the fork returned in a variable named `targetValue` in the `targetEnvironment` after the next top-level command completes. If there is an error in the code, the returned variable will be a try-error. These effects are accomplished via the automatic creation and destruction of a `taskCallback` and other functions inside the `mcpParallelDoManager`. If job results have to be collected before you return to the top level, use [mcpParallelDoCheck](#).

Usage

```
mcpParallelDo(code, targetValue, verbose = TRUE,
      targetEnvironment = .GlobalEnv)
```

Arguments

code	The code to evaluate within a fork wrapped in
targetValue	A character element indicating the variable that the result of that job should be assigned to targetEnvironment
verbose	A boolean element; if TRUE the completion of the fork expr will be accompanied by a message
targetEnvironment	The environment in which you want targetValue to be created

Value

The variable name of the job, this can be manually collected via mcollect or, if on Windows, an empty string

Examples

```
## Create data
data(ToothGrowth)
## Trigger mcpParallelDo to perform analysis on a fork
mcpParallelDo({glm(len ~ supp * dose, data=ToothGrowth)},"interactionPredictorModel")
## Do other things
binaryPredictorModel <- glm(len ~ supp, data=ToothGrowth)
gaussianPredictorModel <- glm(len ~ dose, data=ToothGrowth)
## The result from mcpParallelDo returns in your targetEnvironment,
## e.g. .GlobalEnv, when it is complete with a message (by default)
summary(interactionPredictorModel)

# Example of not returning a value until we return to the top level
for (i in 1:10) {
  if (i == 1) {
    mcpParallelDo({2+2}, targetValue = "output")
  }
  if (exists("output")) print(i)
}

# Example of getting a value without returning to the top level
for (i in 1:10) {
  if (i == 1) {
    mcpParallelDo({2+2}, targetValue = "output")
  }
  mcpParallelDoCheck()
  if (exists("output")) print(i)
}
```

Description

Forces a check on all mcparallelDo() jobs and returns their values to the target environment if they are complete.

Usage

```
mcparallelDoCheck()
```

Value

A named logical vector, TRUE if complete, FALSE if not complete, and an empty logical vector if on Windows

mcparallelDoManagerClass

The mcparallelDoManager Class and Object

Description

The mcparallelDoManager Class and Object

Usage

```
mcparallelDoManagerClass
```

Format

```
Class 'R6ClassGenerator' <mcparallelDoManager> object generator
Public:
  h: list
  runningJobs: list
  addJob: function (jobName, targetValue, verbose, targetEnvironment)
  removeJob: function (x)
  checkJobs: function ()
  clone: function (deep = FALSE)
Parent env: <environment: namespace:mcparallelDo>
Locked objects: TRUE
Locked class: FALSE
Portable: TRUE
- attr(*, "name")= chr "mcparallelDoManager_generator"
```

Index

* **datasets**

- `mcpipelineDoManagerClass`, [5](#)
- `checkIfJobStillRunning`, [2](#)
- `jobCompleteSelfDestructingHandler`, [3](#)
- `mcpipelineDo`, [3](#)
- `mcpipelineDo-package`, [2](#)
- `mcpipelineDoCheck`, [3](#), [4](#)
- `mcpipelineDoManager`
 - `(mcpipelineDoManagerClass)`, [5](#)
- `mcpipelineDoManagerClass`, [5](#)