

# Package: injectoR (via r-universe)

August 21, 2024

**Version** 0.2.4

**Date** 2015-11-25

**Title** R Dependency Injection

**Author** Lev Kuznetsov

**Maintainer** Lev Kuznetsov <levk@jimmy.harvard.edu>

**Depends** R (>= 3.1.0)

**Suggests** testthat

**Description** R dependency injection framework. Dependency injection allows a program design to follow the dependency inversion principle. The user delegates to external code (the injector) the responsibility of providing its dependencies. This separates the responsibilities of use and construction.

**License** LGPL (>= 3)

**URL** <https://github.com/dfci-cccb/injectoR>

**BugReports** <https://github.com/dfci-cccb/injectoR/issues>

**RoxygenNote** 5.0.1

**NeedsCompilation** no

**Date/Publication** 2015-11-30 08:17:48

**Repository** <https://lev-kuznetsov.r-universe.dev>

**RemoteUrl** <https://github.com/cran/injectoR>

**RemoteRef** HEAD

**RemoteSha** 22cd793ec292a7db0962996d4ddb2169d9bcbffa

## Contents

binder	2
default	2
define	3
inject	3

injectoR . . . . .	4
multibind . . . . .	4
shim . . . . .	5
singleton . . . . .	6

<b>Index</b>	<b>7</b>
--------------	----------

---

binder	<i>Binder factory</i>
--------	-----------------------

---

### Description

Binder factory

### Usage

```
binder(parent = .binder, callback = function(binder) binder)
```

### Arguments

parent	of the new binder, injection will propagate up the parent stack looking for keys; if omitted defaults to root binder
callback	called with the newly created binder and the result is returned; if omitted just the new binder is returned

### Value

result of the injected callback if one is specified, otherwise the new binder

### Examples

```
b <- binder ()
```

---

default	<i>Default scope, bindings are provisioned each time a bean is injected</i>
---------	---

---

### Description

Default scope, bindings are provisioned each time a bean is injected

### Usage

```
default(provider)
```

### Arguments

provider	unscoped delegate, no argument function responsible for provision
----------	---

---

define	<i>Creates a key to factory binding</i>
--------	---

---

**Description**

Creates a key to factory binding

**Usage**

```
define(..., scope = default, binder = .binder)
```

**Arguments**

...	injectable bean identifier to factory mappings, the key is the name is matched to a parameter name during injection, the factory responsible for provisioning of the bean, a factory may accept any number of arguments in which case the framework will attempt to inject the argument if a binding to the parameter name exists; if it does not, that argument will not be injected, in which case it is the factory's responsibility to deal with a missing argument
scope	of the bean, wraps the injected factory call specifying provisioning strategy, if omitted a new bean instance will be provisioned each time injection is requested; injectoR also ships with with the singleton scope which will provide once and cache the bean for subsequent calls. Interface allows for custom scoping, the scope parameter must be a function accepting key (name) and the provider - the wrapped injected factory call - a function accepting no parameters responsible for actual provisioning
binder	for this binding, if omitted the new binding is added to the root binder

**Examples**

```
define (hello = function () 'world', binder = binder ())
```

---

inject	<i>Injects the callback function</i>
--------	--------------------------------------

---

**Description**

Injects the callback function

**Usage**

```
inject(callback, binder = .binder)
```

**Arguments**

callback	function to inject, a function accepting arguments to be matched to injectable keys; no errors are thrown if no binding is found for a key, this is the intended mechanic for optional injection, if the callback is able to deal with a missing argument the argument becomes optional
binder	containing the injectables, defaults to root binder if omitted

**Value**

result of the injected callback evaluation

**Examples**

```
inject (function (two) two, define (two = function () 2, binder = binder ()))
inject (function (power) power (2, 4),
        define (power = function (power) function (x, n) if (n < 1) 1 else x * power (x, n - 1)))
inject (function (fibonacci) fibonacci (8),
        define (fibonacci = function (fibonacci)
                function (n) if (n < 3) 1
                            else fibonacci (n - 1) + fibonacci (n - 2), binder = binder ()))
```

---

injector	<i>Dependency injection framework</i>
----------	---------------------------------------

---

**Description**

Dependency injection framework

**Author(s)**

levk

---

multibind	<i>Aggregates multiple factories under one key</i>
-----------	--

---

**Description**

Aggregates multiple factories under one key

**Usage**

```
multibind(key, scope = default, combine = function(this, parent)
          base::c(this, parent), binder = .binder)
```

**Arguments**

key	injectable bean identifier
scope	of the bean, wraps the injected factory call specifying provisioning strategy, if omitted a new bean instance will be provisioned each time injection is requested; injectoR also ships with with the singleton scope which will provide once and cache the bean for subsequent calls. Interface allows for custom scoping, the scope parameter must be a function accepting key (name) and the provider - the wrapped injected factory call - a function accepting no parameters responsible for actual provisioning
combine	aggregation procedure for combination of context and inherited values, a function accepting a list of injectable values from the current binder context and a no argument function to retrieve values of the parent context; if omitted will the binding will aggregate all values
binder	for this binding, if omitted the binding is added to the root binder

**Value**

a function accepting one or more factories for adding elements to the binding; naming the factories will result in named values injected; optionally accepts a scope for the bindings, if omitted defaults to provide on injection; please be aware that the scope is called without key for unnamed multibinding

**Examples**

```
multibind ('keys', binder = binder ()) (function () 'skeleton')
```

---

shim

*Shims libraries*


---

**Description**

Shims libraries

**Usage**

```
shim(..., library.paths = .libPaths(), callback = function() binder,
      binder = .binder)
```

**Arguments**

... zero or more library names to shim binding each exported variable to the binder; if a library name is specified in a named list format (for example `shim(s4='stats4',callback=function(s4.AI` all exported variable names from that library will be prepended with that name and a dot (as in the example); if a library cannot be loaded, no bindings are created for that library and no errors are thrown (but there is an error to console as reported by `requireNamespace`)

library.paths	to use for loading namespace
callback	injected for convenience using the binder specified after shim is completed, if omitted the call returns the binder
binder	for this shim

**Value**

result of the callback if specified, binder otherwise

**Examples**

```
shim ('injector', callback = function (inject) inject, binder = binder ())
```

---

singleton	<i>Singleton scope, bindings of this scope are provided once, on initial demand</i>
-----------	---

---

**Description**

Singleton scope, bindings of this scope are provided once, on initial demand

**Usage**

```
singleton(provider)
```

**Arguments**

provider	unscoped delegate, no argument function responsible for provision
----------	---

**Examples**

```
define (three = function () 3, scope = singleton, binder = binder ())
```

# Index

[binder](#), [2](#)

[default](#), [2](#)

[define](#), [3](#)

[inject](#), [3](#)

[injector](#), [4](#)

[injector-package \(injector\)](#), [4](#)

[multibind](#), [4](#)

[shim](#), [5](#)

[singleton](#), [6](#)