

# Package ‘datamap’

February 14, 2012

**Type** Package

**Title** A system for mapping foreign objects to R variables and environments

**Version** 0.1-1

**Date** 2009-12-03

**Author** Jeffrey Horner <jeffrey.horner@gmail.com>

**Maintainer** Jeffrey Horner <jeffrey.horner@gmail.com>

**Description** datamap utilizes variable bindings and objects of class “UserDefinedDatabase” to provide a simple mapping system to foreign objects. Maps can be used as environments or attached to the search path, and changes to either are persistent. Mapped foreign objects are fetched in real-time and are never cached by the mapping system.

**Depends** DBI

**License** GPL-2

**LazyLoad** yes

**Repository** CRAN

**Date/Publication** 2009-12-03 17:38:05

## R topics documented:

install . . . . .	2
mapAttach . . . . .	2
Mappers . . . . .	3
newMap . . . . .	3
newMapper . . . . .	5
uninstall . . . . .	6

<b>Index</b>	<b>8</b>
--------------	----------

---

install	<i>Function for creating bindings to named foreign objects in a datamap object</i>
---------	--

---

**Description**

install creates the binding for the named foreign object in the datamap object. The foreign object is not created.

**Usage**

```
install( symbols, map )
```

**Arguments**

symbols	A non-zero length character vector naming the bindings to be created in the map.
map	A datamap object.

**See Also**

[uninstall](#)

---

mapAttach	<i>Attach a datamap object to the search path</i>
-----------	---

---

**Description**

Creates a user defined database object and attaches it to the search path. Arguments are the same as attach.

**Usage**

```
mapAttach(map, pos=2, name=NULL, warn.conflicts=TRUE)
```

**Arguments**

map	datamap object.
pos	integer specify where to attach map on search path.
name	name to use for the attached database. If name is missing, then it defaults to 'datamap:MapType' where MapType is the type of the datamap object.
warn.conflicts	see attach.

**Value**

See attach.

**See Also**[attach](#)

---

**Mappers***Global list of Mappers*

---

**Description**

Global list of datamap mappers.

**See Also**[newMapper](#)

---

**newMap***Function for creating datamap objects*

---

**Description**

A datamap is an object database for accessing and storing foreign objects. `newMap` creates the object based on the provided mapper type.

**Usage**

```
newMap( type=character(), ... )
```

**Arguments**

<code>type</code>	Character vector length one describing the mapper to use.
<code>...</code>	Arguments to be passed to the mapper's <code>.init()</code> function. They MUST be named arguments, i.e. <code>name=val</code> .

**Value**

An object of class `codedatamap`.

**See Also**[newMapper](#)

**Examples**

```

newMapper(
  type="EXAMPLE",
  init=function(map,symbols=c('foo','bar','baz'),len=3){

  # Install symbols that the users passes in from newMap().
  lapply(symbols,install,map)

  # Now let's add some state to the internal portion of our map.
  map$len <- len

  # Returning FALSE means failure
  return(TRUE)
},
  get = function(x) {
    cat("I'll just get",x,"for you.\n")

  # len is pulled from the internal portion of the map
  # by lexical scoping rules. Anything can be returned here, but we
  # default to a numeric value
  rnorm(len)
  },
  assign = function(x,val){
    cat("Calling assign",val,"to",x,".\n")
  },
  finalize = function(map){
    cat("Finalization can clear any state, like shutting down database\n")
    cat("connections, socket connections, etc.\n")
  },

  # The rest of the arguments are copied to the internal portion of the map.
  foo = 'bar'
)

m <- newMap('EXAMPLE')

# Summary of the map
m

# [[ works
m[['bar']]

# datamaps are environments
with(m,bar)

# use functions to access either installed objects
# or those that aren't.
m$get('bar')

# adding extra variables to the map.
with(m,x <- 'buzzle')

```

```

m

# attach the map the search path
# and update either the map or the search path position.
# changes are persistent
mapAttach(m)
baz
rm(m)
foo
detach('datamap:EXAMPLE')

```

---

newMapper

*Function for creating new mapper types*


---

### Description

A mapper is a collection of functions that define how a type of foreign object is accessed. `newMapper` creates a new mapper object, assigns it a type, and places it in the global Mapper database.

### Usage

```
newMapper( type=NULL, init=NULL, get=NULL, assign=NULL, finalize=NULL, ...)
```

### Arguments

<code>type</code>	Character vector length one describing the unique type of foreign objects available through the new mapper.
<code>init</code>	A function whose signature is <code>.init(map, ...)</code> where <code>map</code> is a new map object and <code>...</code> are those from the <code>newMap</code> call. Returns <code>TRUE</code> for a successful initialization of the map object or <code>FALSE</code> on error.
<code>get</code>	A function whose signature is <code>.get(symbol)</code> where <code>symbol</code> is a character vector of length one defining the name of the foreign object to fetch. Returns the foreign object or <code>UnboundValue()</code> . <code>NULL</code> is also an acceptable value for <code>.get</code> .
<code>assign</code>	A function whose signature is <code>.assign(symbol, val)</code> where <code>symbol</code> is a character string of length one defining the name of the foreign object to assign to and <code>val</code> is the value the foreign object will take. There are no requirements on the return value of <code>.assign</code> . <code>NULL</code> is also an acceptable value for <code>.assign</code> .
<code>finalize</code>	A function whose signature is <code>'finalize(map)'</code> where <code>map</code> is the object to finalize. There are no requirements on <code>.finalize</code> 's return value. <code>NULL</code> is also an acceptable value.
<code>...</code>	Additional objects that are also added to the mapper environment. They <b>MUST</b> be named arguments, i.e. <code>name=val</code> .

### Value

Invisibly returns the new mapper, an object of class `'dataMapper'`. `newMapper` is called for its side effect of adding the new mapper to the global Mapper database.

**See Also**[newMap](#)**Examples**

```

# Complete example mapper
newMapper(
  type="EXAMPLE",
  init=function(map,symbols=c('foo','bar','baz'),len=3){

# Install symbols that the users passes in from newMap().
  lapply(symbols,install,map)

# Now let's add some state to the internal portion of our map.
  map$len <- len

# Returning FALSE means failure
  return(TRUE)
},
  get = function(x) {
    cat("I'll just get",x,"for you.\n")

# len is pulled from the internal portion of the map
# by lexical scoping rules. Anything can be returned here, but we
# default to a numeric value
    rnorm(len)
  },
  assign = function(x,val){
    cat("Calling assign",val,"to",x,".\n")
  },
  finalize = function(map){
    cat("Finalization can clear any state, like shutting down database\n")
    cat("connections, socket connections, etc.\n")
  },

# The rest of the arguments are copied to the internal portion of the map.
  foo = 'bar'
)

```

---

**uninstall***Function for removing named foreign objects from a datamap object*

---

**Description**

`uninstall` removes the binding for the named foreign object from the datamap object. The foreign object is not deleted.

**Usage**

```
uninstall( symbols, map )
```

**Arguments**

symbols	A non-zero length character vector naming the bindings to be removed from the map.
map	A datamap object.

**See Also**

[install](#)

# Index

## \*Topic **environment**

- install, [2](#)
- mapAttach, [2](#)
- Mappers, [3](#)
- newMap, [3](#)
- newMapper, [5](#)
- uninstall, [6](#)

## \*Topic **programming**

- install, [2](#)
- mapAttach, [2](#)
- Mappers, [3](#)
- newMap, [3](#)
- newMapper, [5](#)
- uninstall, [6](#)

attach, [3](#)

install, [2](#), [7](#)

mapAttach, [2](#)  
Mappers, [3](#)

newMap, [3](#), [6](#)  
newMapper, [3](#), [5](#)

uninstall, [2](#), [6](#)