Package 'datamap'

February 19, 2015

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

NeedsCompilation yes

R topics documented:

---

install	*Function for creating bindings to named foreign objects in a datamap object*

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	*Attach a datamap object to the search path*

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

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

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

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

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