

# Package ‘filehash’

April 17, 2009

**Version** 2.0-1

**Date** 2008-12-19

**Depends** R (>= 2.7.0), methods

**Imports** utils

**Collate** filehash.R filehash-DB1.R filehash-RDS.R coerce.R dump.R hash.R queue.R stack.R zzz.R

**Title** Simple key-value database

**Author** Roger D. Peng <rpeng@jhsph.edu>

**Maintainer** Roger D. Peng <rpeng@jhsph.edu>

**Description** Simple key-value database

**License** GPL (>= 2)

**Repository** CRAN

**Date/Publication** 2008-12-21 16:35:33

## R topics documented:

createQ . . . . .	2
createS . . . . .	2
dbInit . . . . .	3
dbLoad . . . . .	4
dumpObjects . . . . .	6
filehash-class . . . . .	7
filehashFormats . . . . .	9
filehashOption . . . . .	9
queue-class . . . . .	10
stack-class . . . . .	11
stackqueue . . . . .	12
<b>Index</b>	<b>13</b>

`createQ`*Create/Initialize Queue*

---

**Description**

Create or initialize a queue data structure using `filehash` databases

**Usage**

```
createQ(filename)
initQ(filename)
```

**Arguments**

`filename`      character, file name for storing the queue data structure

**Details**

A new queue can be created using `createQ`, which creates a file for storing the queue information and returns an object of class "queue".

**Value**

The `createQ` and `initQ` functions both return an object of class "queue".

**Author(s)**

Roger D. Peng (rpeng@jhsph.edu)

---

`createS`*Create/Initialize Stack*

---

**Description**

Create or initialize a stack data structure using `filehash` databases

**Usage**

```
createS(filename)
initS(filename)
```

**Arguments**

`filename`      character, file name for storing the stack data structure

**Details**

A new stack can be created using `createS`, which creates a file for storing the stack information and returns an object of class "stack".

**Value**

The `createS` and `initS` functions both return an object of class "stack".

**Author(s)**

Roger D. Peng (rpeng@jhsph.edu)

---

 dbInit

*Simple file-based hash table*


---

**Description**

Interface for creating and initializing a simple file-based hash table

**Usage**

```
dbCreate(db, ...)
dbInit(db, ...)
dbReconnect(db, ...)

## S4 method for signature 'ANY':
dbCreate(db, type = NULL, ...)
## S4 method for signature 'ANY':
dbInit(db, type = NULL, ...)
## S4 method for signature 'filehashDB1':
dbReconnect(db, ...)
```

**Arguments**

db	name of database or a database object
type	type of database format. If missing, the default type will be used
...	other arguments passed to methods

**Details**

`dbCreate` creates the necessary files or directory for the database. If those files already exist nothing is done.

`dbInit` takes a database name and returns an object inheriting from class "filehash".

The `type` argument specifies the format in which the database should be stored on the disk. If not specified, the default type will be used (as specified by `filehashOption`).

**Value**

dbCreate returns TRUE upon success and FALSE in the event of an error. dbInit returns an object inheriting from class "filehash"

**Note**

The function dbInitialize has been deprecated. Use dbInit instead.

**Author(s)**

Roger D. Peng

**See Also**

See [filehash-class](#) more information and examples and [filehashOption](#) for setting the default database type.

---

dbLoad

*Load database into environment*

---

**Description**

Load entire database into an environment

**Usage**

```
db2env(db)
dbLoad(db, ...)
dbLazyLoad(db, ...)

## S4 method for signature 'filehash':
dbLoad(db, env = parent.frame(2), keys = NULL, ...)
## S4 method for signature 'filehash':
dbLazyLoad(db, env = parent.frame(2), keys = NULL, ...)
```

**Arguments**

db	database object
env	an environment
keys	character vector of database keys to load
...	other arguments passed to methods

## Details

`db2env` loads the entire database `db` into an environment via calls to `makeActiveBinding`. Therefore, the data themselves are not stored in the environment, but a function pointing to the data in the database is stored. When an element of the environment is accessed, the function is called to retrieve the data from the database. If the data in the database is changed, the changes will be reflected in the environment.

`dbLoad` loads objects in the database directly into the environment specified, like `load` does except with active bindings. `dbLoad` takes a second argument `env`, which is an environment, and the default for `env` is `parent.frame()`.

The use of `makeActiveBinding` in `db2env` and `dbLoad` allows for potentially large databases to, at least conceptually, be used in R, as long as you don't need simultaneous access to all of the elements in the database.

With `dbLazyLoad` database objects are "lazy-loaded" into the environment. Promises to load the objects are created in the environment specified by `env`. Upon first access, those objects are copied into the environment and will from then on reside in memory. Changes to the database will not be reflected in the object residing in the environment after first access. Conversely, changes to the object in the environment will not be reflected in the database. This type of loading is useful for read-only databases.

## Value

For `db2env`, an environment is returned, the elements of which are the keys of the database. For `dbLoad` and `dbLazyLoad`, a character vector is returned (invisibly) containing the keys associated with the values loaded into the environment.

## Author(s)

Roger D. Peng

## See Also

[dbInit](#) and [filehash-class](#)

## Examples

```
dbCreate("myDB")
db <- dbInit("myDB")
dbInsert(db, "a", rnorm(100))
dbInsert(db, "b", 1:10)

env <- db2env(db)
ls(env) ## "a", "b"
print(env$b)
mean(env$a)
env$a <- rnorm(100)
mean(env$a)

env$b[1:5] <- 5:1
print(env$b)
```

```

env <- new.env()
dbLoad(db, env)
ls(env)

env <- new.env()
dbLazyLoad(db, env)
ls(env)

```

---

dumpObjects

*Dump objects of database*


---

### Description

Dump R objects to a filehash database

### Usage

```

dumpObjects(..., list = character(0), dbName, type = NULL, envir = parent.frame())
dumpImage(dbName = "Rworkspace", type = NULL)
dumpDF(data, dbName = NULL, type = NULL)
dumpList(data, dbName = NULL, type = NULL)
dumpEnv(env, dbName)

```

### Arguments

<code>...</code>	R objects to dump
<code>list</code>	character vector of names of objects to dump
<code>dbName</code>	character, name of database to which objects should be dumped
<code>type</code>	type of database to create
<code>envir</code>	environment from which to obtain objects
<code>data</code>	a data frame or a list
<code>env</code>	an environment

### Details

Objects dumped to a database can later be loaded via `dbLoad` or can be accessed with `dbFetch`, `dbList`, etc. Alternatively, the `with` method can be used to evaluate code in the context of a database. If a database with name `dbName` already exists, objects will be inserted into the existing database (and values for already-existing keys will be overwritten).

`dumpDF` is different in that each variable in the data frame is stored as a separate object in the database. So each variable can be read from the database separately rather than having to load the entire data frame into memory. `dumpList` works in a similar way.

The `dumpEnv` function takes an environment and stores each element of the environment in a filehash database.

**Value**

An object of class "filehash" is returned and a database is created.

**Author(s)**

Roger D. Peng

**Examples**

```
data <- data.frame(y = rnorm(100), x = rnorm(100), z = rnorm(100))
db <- dumpDF(data, dbName = "dataframe.dump")
fit <- with(db, lm(y ~ x + z))
summary(fit)

db <- dumpList(list(a = 1, b = 2, c = 3), "list.dump")
db$a
```

---

filehash-class      *Class "filehash"*

---

**Description**

These functions form the interface for a simple file-based key-value database (i.e. hash table).

**Objects from the Class**

Objects can be created by calls of the form `new("filehash", ...)`.

**Slots**

**name:** Object of class "character", name of the database.

**Additional slots for "filehashDB1"**

**datafile:** full path to the database file.

**meta:** list containing an environment for database metadata.

**Additional slots for "filehashRDS"**

**dir:** Directory where files are stored.

## Methods

**dbDelete** The `dbDelete` function is for deleting elements, but for the "DB1" format all it does is remove the key from the lookup table. The actual data are still in the database (but inaccessible). If you reinsert data for the same key, the new data are simply appended on to the end of the file. Therefore, it's possible to have multiple copies of data lying around after a while, potentially making the database file big. The "RDS" format does not have this problem.

**dbExists** check to see if a key exists.

**dbFetch** retrieve the value associated with a given key.

**dbMultiFetch** retrieve values associated with multiple keys (a list of those values is returned).

**dbInsert** insert a key-value pair into the database. If that key already exists, its associated value is overwritten. For "RDS" type databases, there is a `safe` option (defaults to `TRUE`) which allows the user to insert objects somewhat more safely (objects should not be lost in the event of an interrupt).

**dbList** list all keys in the database.

**dbReorganize** The `dbReorganize` function is there for the purpose of rewriting the database to remove all of the stale entries. Basically, this function creates a new copy of the database and then overwrites the old copy. This function has not been tested extensively and so should be considered *experimental*. `dbReorganize` is not needed when using the "RDS" format.

**dbUnlink** delete an entire database from the disk

**show** print method

**with** allows `with` to be used with "filehash" objects much like it can be used with lists or data frames

`[[,][<-` elements of a database can be accessed using the `[[` operator much like a list or environment, but only character indices are allowed

`,<-` elements of a database can be accessed using the `$` operator much like with a list or environment

**lapply** works much like `lapply` with lists; a list is returned.

## Author(s)

Roger D. Peng ([rpeng@jhsph.edu](mailto:rpeng@jhsph.edu))

## Examples

```
dbCreate("myDB") ## Create database 'myDB'
db <- dbInit("myDB")
dbInsert(db, "a", 1:10)
dbInsert(db, "b", rnorm(1000))
dbExists(db, "b") ## 'TRUE'

dbList(db) ## c("a", "b")
dbDelete(db, "a")
dbList(db) ## "b"

with(db, mean(b))
```

---

filehashFormats      *List and register filehash formats*

---

**Description**

List and register filehash backend database formats.

**Usage**

```
registerFormatDB(name, funlist)
filehashFormats(...)
```

**Arguments**

name	character, name of database format
funlist	list of functions for creating and initializing a database format
...	list of functions for registering a new database format

**Details**

registerFormatDB can be used to register new filehash backend database formats. filehashFormats called with no arguments lists information on available formats.

**Value**

filehashFormats returns a list containing information on the available filehash formats.

---

filehashOption      *Set filehash options*

---

**Description**

Set global filehash options

**Usage**

```
filehashOption(...)
```

**Arguments**

...	name-value pairs for options
-----	------------------------------

**Details**

Currently, the only option that can be set is the default database type (defaultType) which can be "DB1", "RDS" or "DB".

**Value**

`filehashOptions` returns a list of current settings for all options.

**Author(s)**

Roger D. Peng

---

queue-class

*Class "queue"*

---

**Description**

A queue implementation using a `filehash` database

**Objects from the Class**

Objects can be created by calls of the form `new("queue", ...)` or by calling `createQ`. Existing queues can be initialized with `initQ`.

**Slots**

**queue:** Object of class `"filehashDB1"`

**name:** Object of class `"character"`: the name of the queue (default is the file name in which the queue data are stored)

**Methods**

**isEmpty** signature `(db = "queue")`: returns TRUE/FALSE depending on whether there are elements in the queue.

**pop** signature `(db = "queue")`: returns the value of the "top" (i.e. head) of the queue and subsequently removes that element from the queue; an error is signaled if the queue is empty

**push** signature `(db = "queue")`: adds an element to the tail ("bottom") of the queue

**show** signature `(object = "queue")`: prints the name of the queue

**top** signature `(db = "queue")`: returns the value of the "top" (i.e. head) of the queue; an error is signaled if the queue is empty

**Author(s)**

Roger D. Peng (rpeng@jhsph.edu)

**Examples**

```
showClass("queue")
```

---

stack-class	Class "stack"
-------------	---------------

---

### Description

A stack implementation using a filehash database

### Objects from the Class

Objects can be created by calls of the form `new("stack", ...)` or by calling `createS`. Existing queues can be initialized with `initS`.

### Slots

**stack:** Object of class "filehashDB1"

**name:** Object of class "character": the name of the stack (default is the file name in which the stack data are stored)

### Methods

**isEmpty** signature (db = "stack"): returns TRUE/FALSE depending on whether there are elements in the stack.

**pop** signature (db = "stack"): returns the value of the top of the stack and subsequently removes that element from the stack; an error is signaled if the stack is empty

**push** signature (db = "stack"): adds an element to the top of the stack

**show** signature (object = "stack"): prints the name of the stack

**top** signature (db = "stack"): returns the value of the top of the stack; an error is signaled if the stack is empty

**mpush** signature (db = "stack"): works like push except it can push multiple objects in a list on to the stack

### Author(s)

Roger D. Peng (rpeng@jhsph.edu)

### Examples

```
showClass("stack")
```

---

`stackqueue`*Operations on Stacks/Queues*

---

**Description**

Functions for interacting with stack and queue data structures implemented using `filehash` databases.

**Usage**

```
push(db, val, ...)  
mpush(db, vals, ...)  
pop(db, ...)  
top(db, ...)  
isEmpty(db, ...)
```

**Arguments**

<code>db</code>	an object of class "stack" or "queue"
<code>val</code>	an R object
<code>vals</code>	a list of R objects
<code>...</code>	arguments passed to other methods

**Details**

Note that for `mpush`, if `vals` is not a list it will be coerced to a list via `as.list`. Currently, `mpush` is only implemented for "stack"s.

**Value**

`push` and `mpush` return nothing useful; `pop` returns a value from the stack/queue and deletes that value from the stack/queue; `top` returns the "top" value from the stack/queue; `isEmpty` returns TRUE/FALSE depending on whether the stack/queue is empty or not. Both `pop` and `top` signal an error if the stack/queue is empty.

**Author(s)**

Roger D. Peng ([rpeng@jhsph.edu](mailto:rpeng@jhsph.edu))

# Index

## \*Topic **classes**

filehash-class, 7  
queue-class, 9  
stack-class, 10

## \*Topic **database**

createQ, 1  
createS, 2  
dbInit, 3  
dbLoad, 4  
dumpObjects, 5  
filehashOption, 9  
stackqueue, 11

## \*Topic **utilities**

filehashFormats, 8

[, filehash, ANY, ANY, missing-method  
(filehash-class), 7  
[, filehashDB1, character, missing, missing-method  
(filehash-class), 7  
[[, filehash, character, missing-method  
(filehash-class), 7  
[[, filehash, numeric, missing-method  
(filehash-class), 7  
[[<-, filehash, character, missing-method  
(filehash-class), 7  
[[<-, filehash, numeric, missing-method  
(filehash-class), 7  
\$, filehash, character-method  
(filehash-class), 7  
\$<-, filehash, character-method  
(filehash-class), 7  
  
coerce, filehashDB, filehashDB1-method  
(filehash-class), 7  
coerce, filehashDB, filehashRDS-method  
(filehash-class), 7  
coerce, filehashDB1, filehashRDS-method  
(filehash-class), 7  
coerce, filehashDB1, list-method  
(filehash-class), 7

coerce, filehashRDS, filehashDB-method  
(filehash-class), 7  
createQ, 1  
createS, 2  
  
db2env (dbLoad), 4  
dbCreate (dbInit), 3  
dbCreate, ANY-method (dbInit), 3  
dbDelete (filehash-class), 7  
dbDelete, filehashDB, character-method  
(filehash-class), 7  
dbDelete, filehashDB1, character-method  
(filehash-class), 7  
dbDelete, filehashRDS, character-method  
(filehash-class), 7  
dbExists (filehash-class), 7  
dbExists, filehashDB, character-method  
(filehash-class), 7  
dbExists, filehashDB1, character-method  
(filehash-class), 7  
dbExists, filehashRDS, character-method  
(filehash-class), 7  
dbFetch (filehash-class), 7  
dbFetch, filehashDB, character-method  
(filehash-class), 7  
dbFetch, filehashDB1, character-method  
(filehash-class), 7  
dbFetch, filehashRDS, character-method  
(filehash-class), 7  
dbInit, 3, 5  
dbInit, ANY-method (dbInit), 3  
dbInitialize (dbInit), 3  
dbInsert (filehash-class), 7  
dbInsert, filehashDB, character-method  
(filehash-class), 7  
dbInsert, filehashDB1, character-method  
(filehash-class), 7  
dbInsert, filehashRDS, character-method  
(filehash-class), 7  
dbLazyLoad (dbLoad), 4

- dbLazyLoad, filehash-method  
(*dbLoad*), 4
- dbList (*filehash-class*), 7
- dbList, filehashDB-method  
(*filehash-class*), 7
- dbList, filehashDB1-method  
(*filehash-class*), 7
- dbList, filehashRDS-method  
(*filehash-class*), 7
- dbLoad, 4
- dbLoad, filehash-method (*dbLoad*), 4
- dbMultiFetch (*filehash-class*), 7
- dbMultiFetch, filehashDB1, character-method  
(*filehash-class*), 7
- dbMultiFetch, filehashDB1-method  
(*filehash-class*), 7
- dbReconnect (*dbInit*), 3
- dbReconnect, filehashDB1-method  
(*dbInit*), 3
- dbReorganize (*filehash-class*), 7
- dbReorganize, filehashDB-method  
(*filehash-class*), 7
- dbReorganize, filehashDB1-method  
(*filehash-class*), 7
- dbUnlink (*filehash-class*), 7
- dbUnlink, filehashDB-method  
(*filehash-class*), 7
- dbUnlink, filehashDB1-method  
(*filehash-class*), 7
- dbUnlink, filehashRDS-method  
(*filehash-class*), 7
- dumpDF (*dumpObjects*), 5
- dumpEnv (*dumpObjects*), 5
- dumpImage (*dumpObjects*), 5
- dumpList (*dumpObjects*), 5
- dumpObjects, 5
  
- filehash-class, 4, 5
- filehash-class, 7
- filehashDB-class  
(*filehash-class*), 7
- filehashDB1-class  
(*filehash-class*), 7
- filehashFormats, 8
- filehashOption, 4, 9
- filehashRDS-class  
(*filehash-class*), 7
  
- initQ (*createQ*), 1
  
- initS (*createS*), 2
- isEmpty (*stackqueue*), 11
- isEmpty, queue-method  
(*queue-class*), 9
- isEmpty, stack-method  
(*stack-class*), 10
  
- lapply, filehash-method  
(*filehash-class*), 7
  
- mpush (*stackqueue*), 11
- mpush, stack-method (*stack-class*),  
10
  
- pop (*stackqueue*), 11
- pop, queue-method (*queue-class*), 9
- pop, stack-method (*stack-class*), 10
- push (*stackqueue*), 11
- push, queue-method (*queue-class*), 9
- push, stack-method (*stack-class*),  
10
  
- queue-class, 9
  
- registerFormatDB  
(*filehashFormats*), 8
  
- show, filehash-method  
(*filehash-class*), 7
- show, queue-method (*queue-class*), 9
- show, stack-method (*stack-class*),  
10
  
- stack-class, 10
- stackqueue, 11
  
- top (*stackqueue*), 11
- top, queue-method (*queue-class*), 9
- top, stack-method (*stack-class*), 10
  
- with, filehash-method  
(*filehash-class*), 7