Package 'hash'

March 22, 2022

Type Package

Title Full Featured Implementation of Hash Tables/Associative Arrays/Dictionaries

Version 2.2.6.2

Date 2022-03-21

Author Christopher Brown <chris.brown@decisionpatterns.com>

Maintainer John Hughes <drjphughesjr@gmail.com>

Depends R (>= 2.12.0), methods, utils

Suggests testthat

Description Implements a data structure similar to hashes in Perl and dictionaries in Python but with a purposefully R flavor. For objects of appreciable size, access using hashes outperforms native named lists and vectors.

License GPL (>= 2)

URL http://www.johnhughes.org

NeedsCompilation no

Repository CRAN

Date/Publication 2022-03-22 13:10:05 UTC

R topics documented:

`hash-package` *Hash/associative array/dictionary data structure for the R language.*

Description

This S4 class is designed to provide a hash-like data structure in a native R style and provides the necessary methods for all general methods for hash operations.

Details

Package:	hash
Type:	Package
Version:	2.2.6
Date:	2013-02-20
License:	GPL (>= 2)
LazyLoad:	yes
Depends:	R (>= 2.12.0), utils, methods

Note

R is slowly moving toward a native implementation of hashes using enviroments, (cf. `Extract`. Access to environments using $ and [[has been available for some time and recently objects can inherit from environments, etc. But many features that make hashes/dictionaries great are still lacking, such as the slice operation, [.

The hash package is the only full featured hash implementation for the R language. It provides more features and finer control of the hash behavior than the native feature set and has similar and sometimes better performance.

HASH KEYS must be a valid character value and may not be the empty string `""`.

HASH VALUES can be any R value, vector or object.

PASS-BY REFERENCE. Environments and hashes are special objects in R because only one copy exists globally. When provide as an argument to a function, no local copy is made and any changes to the hash in the functions are reflected globally.

PERFORMANCE. Hashes are based on R's native environments and are designed to be exceedingly fast using the environments internal hash table. For small data structures, a list will out-perform a hash in nearly every case. For larger data structure, i.e. > 500 key value pair the performance of the hash becomes faster. Much beyond that the performance of the hash far outperforms native lists.

MEMORY. Objects of class `hash` do not release memory with a call to `rm`. `clear` must be called before `rm` to properly release the memory.

Author(s)

Christopher Brown

Maintainer: Christopher Brown <chris.brown -at- decisionpatterns -dot- com>

References

http://www.mail-archive.com/r-help@r-project.org/msg37637.html

http://www.mail-archive.com/r-help@r-project.org/msg37650.html

http://tolstoy.newcastle.edu.au/R/help/05/12/index.html\#18192

See Also

See also `hash` , `hash-accessors` and `environment`

Examples

```
h <- hash( keys=letters, values=1:26 )
h <- hash( letters, 1:26 )

h$a # 1

h$foo <- "bar"
h[ "foo" ]
h[[ "foo" ]]

clear(h)
rm(h)
```

`.set` *assign key-value pair(s) to a hash*

Description

`.set` is an internal method for assigning key-value pairs to a `hash`. Normally, there is no need to use this function. Convenient access is provided by: `hash,\$,[` and `[[` and their corresponding replacement methods.

`.set` takes 4 types of arguments: explicitly named key and value vectors named key-value pairs named vectors implicit key-value pairs

The keys are automatically coerced to valid keys and are restricted to character classes. Values are free to be any valid R object.

Usage

```
.set( hash, ... )
```

Arguments

`hash`	An hash object on which to set the key-value pair(s)
`...`	Any of several ways to specify keys and values. See Details.

Details

`.set` sets zero or more key-value pairs. If the key(s) already exist, existing values are silently clobbered. Otherwise, a new value is saved for each key. Keys and values are by the `...` argument. If `...` is:

made only of explicitly named `keys` and `values` arguments then these are taken as the keys and values respectively.

a named list, then the names are taken as keys and list elements are taken as values.

a named vector, then the names are taken as keys. Vector elements are taken as values.

of length two, keys are taken from the first element, values from the second.

Keys are coerced to type `character`.

Keys and values are assigned to the hash as follows:

IF `keys` and `values` are the same length, key-value pairs are added to the hash pairwise.

IF `keys` is a vector of length 1, then this key is assigned the entire `values` vector.

IF `values` is a vector of length 1, each key of `keys` is assigned the value given by `values`

IF `keys` and `values` are of different lengths, both greater than one, then the assignment is considered ambiguous and an error is thrown.

Value

`.set` exists solely for its side-effects. An invisible NULL is returned.

Author(s)

Christopher Brown

See Also

See also `hash`, `environment`

Examples

```
h <- hash()

.set( h, keys=letters, values=1:26 )
.set( h, a="foo", b="bar", c="baz" )
.set( h, c( aa="foo", ab="bar", ac="baz" ) )
```

```
clear(h)
.set( h, letters, values )
```

clear *Removes all key-value pairs from a hash*

Description

`clear` removes all key-values from a hash.

Usage

```
clear(x)
```

Arguments

x	A hash object.

Details

Currently `clear` removes (`rm`) the key-value pairs on the hash. For large hashes it might be faster to reinitialize the hash, though this might cause memory leaks.

Value

None. Method clear exists entirely for its side effects.

Note

`clear` should be called prior to removing a hash. This ensures that the memory from the environment is freed.

Author(s)

Christopher Brown

See Also

`del` to remove specific key-values from the hash. `hash`.

Examples

```
h <- hash( letters, 1:26 )
h # An object of type 'hash' containing 26 key-value pairs.
clear(h)
h # An object of type 'hash' containing 0 key-value pairs.
```

copy-methods *Create a seperate copy of a hash object.*

Description

The copy hash method creates a independent copy of a hash object. Creating a copy using the assingment operator, `<-`, does not work as expected, since hashes are based on environments and environments are reference objects in R. The assignment operator consequently creates a linked copy to the original hash and not an independent copy. The `copy` method provides an identical unlinked copy of the hash.

Value

A hash object.

Methods

`signature(x = "hash")` Creates and returns an identical, independent, unreferenced copy of the the hash.

Author(s)

Christopher Brown

See Also

`environment`

Examples

```
h <- hash( a=1, b=2 )
h.new <- copy( h )
```

del *Remove key-value pair(s) from a hash*

Description

Removes key-value pair(s) from a hash.

Usage

```
del(x,hash)
delete(x,hash)
```

Arguments

x	An object that will be coerced to valid key(s) to be removed from the hash. x will be coerced to a valid hash keys using make.keys
hash	A hash object

Value

None. This method exists solely for the side-effects of removing items from the hash.

Author(s)

Christopher Brown

See Also

See Also as hash, make.keys.

Examples

```
h <- hash( letters, 1:26 )
h # 26 elements
del( "a", h )
h # 25 elements
```

Format hash object for pretty printing

Methods for Function format in Package 'hash'

Description

Format a hash for printing.

Methods

x = "hash" Format a hash for pretty printing.

See Also

See also format

`has.key` *Test for existence of key(s) on a hash*

Description

`has.key` returns a logical vector as long as `keys`, indicating which keys are defined on the hash.

Usage

```
has.key(key, hash, ...)
```

Arguments

`key`	A vector whose entries will be coerced to valid keys.
`hash`	A `hash` object.
`...`	arguments passed to further functions

Details

None.

Value

`logical`	A logical vector of length `key` indicating whether the key is defined in the hash. `has.key` also accepts `...` to be passed to underlying `sapply`

Author(s)

Christopher Brown

See Also

See also `hash`

Examples

```
h <- hash( letters, 1:26 )
all( has.key( letters, h ) ) # TRUE
```

hash — *hash/associative array/dictionary data structure for the R language*

Description

Preferred constructor for the `hash-class`.

Usage

```
  hash(...)

  is.hash(x)

  ## S3 method for class 'hash'
as.list(x, all.names = FALSE, ...  )
```

Arguments

`x`	A hash object.
`all.names`	a logical indicating whether to copy all values or (default) only those whose names do not begin with a dot
`...`	Additional arguments passed to the function

Details

`hash` returns a hash object. Key-value pairs may be specified via the `...` argument as explicity arguments `keys` and `values`, as named key-value pairs, as a named vector or as implicit key, value vectors. See examples below for each type.

Keys must be a valid R name, must be a character vector and must not be the empty string, `""`. Values are restricted to any valid R objects.

See `.set` for further details and how key-value vectors of unequal length are interpretted.

Hashes may be accessed via the standard R accessors `[`, `[[` and `\$`. See `hash-accessors` for details.

`is.hash` returns a boolean value indicating if the argument is a hash object.

`as.list.hash` coerces the hash to a list.

Value

For `hash`, an object of class hash.

Author(s)

Christopher Brown

See Also

.set, hash-accessors

Examples

```
hash()

hash( key=letters, values=1:26 )

hash( 1:3, lapply(1:3, seq, 1 ))

hash( a=1, b=2, c=3 )
hash( c(a=1, b=2, c=3) )
hash( list(a=1,b=2,c=3) )

hash( c("foo","bar","baz"), 1:3 )
hash( c("foo","bar","baz"),  lapply(1:3, seq, 1 ) )
hash( letters, 1:26 )

h <- hash( letters, 1:26 )
h$a
h$b
h[[ "a" ]]
h[ letters[1:3] ]

h$a<-100
# h[['a']]<-letters

is.hash(h)
as.list(h)

clear(h)
rm(h)
```

`hash-accessors` *Accessor methods for the hash class.*

Description

R style accesors for the hash-class.

Details

These are the hash accessor methods. They closely follow an R style.

`$` is a look-up operator for a single key. The native `$` method is used. The key is taken as a string literal and is not interpreted.

`[[` is the look-up, extraction operator. It returns the values of a single key.

`[` is a subseting operator. It returns a (sub) hash with the specified keys. All other keys are removed.

Value

\$ and [[return the value for the supplied argument. If a key does not match an existing key, then `NULL` is returned with a warning.

[returns a hash slice, a sub hash with only the defined keys.

Author(s)

Christopher Brown

See Also

`hash`, `values`, `.set`, `as.list`

Examples

```
h <- hash()
h <- hash( letters, 1:26 )

h$a
h$a <- "2"
h$z <- NULL          # Removes 'z' from

h[['a']]
h[['a']] <- 23

h[ letters[1:4] ]    # hash with a,b,c,d
h[ letters[1:4] ] <- 4:1
```

`hash-class` *Class "hash"*

Description

Implements a S4 hash class in R similar to hashes / associatesd arrays / dictionaries in other programming languages. Where possible, the hash class uses the standard R accessors: `\$`, `[` and `[[`. Hash construction is flexible and takes several syntaxes and all hash operations are supported.

For shorter key-value pairs, lists might yield higher performance, but for lists of appreciable length hash objects handly outperform native lists.

Slots

`.xData`**:** Object of class `"environment"`. This is the hashed environment used for key-value storage.

Extends

environment

Methods

HASH ACCESSORS:

`signature(x = "hash", i = "ANY", j = "missing")`: Slice Replacement

[[<-`signature(x = "hash", i = "ANY", j = "missing", drop = "missing")` : Slice

[[<- `signature(x = "hash", i = "ANY", j = "missing")`: Single key replacement with interpolation.

[[`signature(x = "hash", i = "ANY", j = "missing")`: i Single key look-up with interpolation

\$<- `signature(x = "hash")`: Single key replacement no interpolation

\$ `signature(x = "hash")`: Single key lookup no interpolation

Manipulation:

clear `signature(x = "hash")`: Remove all key-value pairs from hash

del `signature(x = "ANY", hash = "hash")`: Remove specified key-value pairs from hash

has.key `signature(key = "ANY", hash = "hash")`: Test for existence of key

is.empty `signature(x = "hash")`: Test if no key-values are assigned

length `signature(x = "hash")`: Return number of key-value pairs from the hash

keys `signature(hash = "hash")`: Retrieve keys from hash

values `signature(x = "hash")`: Retrieve values from hash

copy `signature(x = "hash")`: Make a copy of a hash using a new environment.

format `signature(x = "hash")`: Internal function for displaying hash

Note

HASH KEYS must be a valid character value and may not be the empty string `""`.

HASH VALUES can be any R value, vector or object.

PASS-BY REFERENCE. Environments and hashes are special objects in R because only one copy exists globally. When provide as an argument to a function, no local copy is made and any changes to the hash in the functions are reflected globally.

PERFORMANCE. Hashes are based on environments and are designed to be exceedingly fast using the environments internal hash table. For small data structures, a list will out-perform a hash in nearly every case. For larger data structure, i.e. >100-1000 key value pair the performance of the hash becomes faster. Much beyond that the performance of the hash far outperforms native lists.

MEMORY. Objects of class `hash` do not release memory with a call to `rm`. `clear` must be called before `rm` to properly release the memory.

Author(s)

Christopher Brown

References

http://en.wikipedia.org/wiki/Hash_table

http://en.wikipedia.org/wiki/Associative_array

See Also

hash-accessors, environment.

Examples

```
showClass("hash")
```

---

invert *Create an inverted hash.*

Description

THIS IS AN EXPERIMENTAL FUNCTION. THE IMPLEMENTATION OR INTERFACE MAY CHANGE WITHOUT WARNING.

Invert creates an inverted hash from an existing hash. An inverted hash is one in which the keys and values are exchanged.

Usage

```
invert(x)
inverted.hash(...)
```

Arguments

x	A hash object
...	Arguments passed to the hash function.

Details

For `invert`, keys and value elements switch. Each element of the `values(x)` is coerced to a key. The value becomes the associated key.

For `inverted.hash`, a hash is created than inverted. It is defined as:

`function(...) invert(hash(...))`

Value

A hash object with: keys as the unique elements of `values(x)` and values as the associated `keys{x}`

Author(s)

Christopher Brown

See Also

See also `link{hash}` and make.keys

Examples

```
h <- hash( a=1, b=1:2, c=1:3 )
invert(h)

inverted.hash( a=1, b=1:2, c=1:3 )
```

is.empty *Test if a hash has no key-value pairs.*

Description

`is.empty` tests to see if any key value pairs are assigned on a `hash` object.

Usage

```
is.empty(x)
```

Arguments

`x`	hash object.

Details

Returns `TRUE` if no key-value pairs are defined for the hash, `FALSE` otherwise.

Value

logical.

Author(s)

Christopher Brown.

See Also

`exists`.

Examples

```
h <- hash( a=1, b=2, c=3 )
is.empty(h)    # FALSE
clear(h)
is.empty(h)    # TRUE
h <- hash()
is.empty(h)    # TRUE
```

`keys` *Returns key(s) from a hash*

Description

Returns the key(s) from a hash

Usage

```
  keys(x)

  ## S3 method for class 'hash'
names(x)
```

Arguments

`x`	A `hash` object.

Details

Returns the character vector containing the keys of a hash object.

Value

`keys`	A vector of type character

Author(s)

Christopher Brown

See Also

See Also `hash`.

Examples

```
  h <- hash( letters, 1:26 )
  keys(h)  # letters

  names(h) # same
```

`length` *Returns the number of items in a hash*

Description

Returns the number of items in a hash

Details

Return the number of items in the hash by calling `length` on the internal environment.

Value

`integer`	Number of items in the hash.

Author(s)

Christpher Brown

See Also

See Also `hash`, `length`

Examples

```
h <- hash( letters, 1:26 )
length(h) # 26
```

`make.keys` *creates/coerces objects to proper hash keys*

Description

Given an vector of any type, `make.keys` tries to coerce it into a character vector that can be used as a hash key. This is used internally by the hash package and should not be normally needed.

Usage

```
make.keys(key)
```

Arguments

`key`	An object that represents the key(s) to be coerced to a valid hash keys.

Details

This function is used internally by the hash class to ensure that the keys are valid. There should be no need to use this externally and is only documented for completeness.

Value

A character vector of valid keys

Author(s)

Christopher Brown

See Also

See also as hash

Examples

```
make.keys( letters )
make.keys( 1:26 )
```

---

values *Extract values of a hash object.*

Description

Extract `values` from a `hash` object. This is a pseudo- accessor method that returns hash values (without keys) as a vector if possible, a list otherwise.

simplifies them to the lowest order (c.f. simplify). It is very similar to `h[[ keys(h) ]]`, An optional key. It is identical to `h[[ keys(h) ]]`.

For details about hash accessors, please see hash-class

Usage

```
  ## S4 method for signature 'hash'
values(x, keys=NULL, ...)
  ## S4 replacement method for signature 'hash'
values(keys=NULL) <- value
```

Arguments

`x`	The hash from where the values retrieved
`keys`	A vector of keys to be returned.
`...`	Arguments passed to sapply
`value`	For the replacement method, the value(s) to be set.

Details

The `values` method returns the values from a hash. It is similar to `h[[ keys(h) ]]` except that a named vector or list is returned instead of a hash. : By default, the returned values are simplified by coercing to a vector or matrix if possible; elements are named after the corresponding key. If the values are of different types or of a complex class than a named list is returned. Argument `simplify` can be used to control this behavior.

If a character vector of `keys` is provided, only these keys are returned. This also allows for returning values mulitple times as in:

`values(h,keys=c('a','a','b' ) )`

This is now the preferred method for returning multiple values for the same key.

The replacement method, `values<-` can replace all the values or simply those associated with the supplied `keys`. Use of the accessor '[' is almost always preferred.

Value

Please see details for which value will be returned:

`vector`	Vector with the type as the values of the hash
`list`	list containing the values of the hash

Author(s)

Christopher Brown

References

http://blog.opendatagroup.com/2009/10/21/r-accessors-explained/

See Also

See also `hash`, `sapply`.

Examples

```
h <- hash( letters, 1:26 )
values(h)  # 1:26
values(h, simplify = FALSE )
values(h, USE.NAMES = FALSE )

h <- hash( 1:26, letters )
values(h)
values(h, keys=1:5 )
values(h, keys=c(1,1,1:5) )
values(h, keys=1:5) <- 6:10
values(h) <- rev( letters )
```

Index