

# Package: memofunc (via r-universe)

October 12, 2026

**Type** Package

**Title** Programmable Function Memoization for R

**Version** 1.0.3

**Description** Memoize R functions with explicit, predictable cache behavior. Function calls become reusable memory: repeated computations return quickly, while cache keys remain transparent and deterministic. Keys are derived from function identity and normalized call arguments, so implementation changes naturally invalidate stale values. Includes runtime controls for forced recomputation and dry-run cache checks, plus pluggable storage backends for in-memory, local file, and provider-backed object storage.

**Depends** R (>= 3.5.0)

**License** GPL-3

**URL** <https://github.com/rwetherall/memofunc>,  
<https://rwetherall.github.io/memofunc/>

**BugReports** <https://github.com/rwetherall/memofunc/issues>

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.3.3

**Imports** digest, uuid, magrittr

**Suggests** testthat, devtools, roxygen2, covr, AzureStor

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

**Date/Publication** 2026-04-15 09:04:10 UTC

**RemoteUrl** <https://github.com/rwetherall/memofunc>

**RemoteRef** HEAD

**RemoteSha** 204a9841366d994b94a3d6e3e37eaa3472756100

Contents

|                                |           |
|--------------------------------|-----------|
| functionCall . . . . .         | 2         |
| hash . . . . .                 | 3         |
| hash.default . . . . .         | 4         |
| hash.function . . . . .        | 5         |
| hash.functionCall . . . . .    | 6         |
| hash.list . . . . .            | 7         |
| invalidate . . . . .           | 8         |
| is.memo . . . . .              | 9         |
| memo . . . . .                 | 9         |
| memo.cache . . . . .           | 13        |
| memo.function . . . . .        | 13        |
| storage.clear . . . . .        | 14        |
| storage.clear.memory . . . . . | 16        |
| storage.clear.object . . . . . | 18        |
| storage.get . . . . .          | 20        |
| storage.get.memory . . . . .   | 22        |
| storage.get.object . . . . .   | 24        |
| storage.has . . . . .          | 26        |
| storage.has.memory . . . . .   | 28        |
| storage.has.object . . . . .   | 30        |
| storage.init . . . . .         | 32        |
| storage.init.memory . . . . .  | 34        |
| storage.init.object . . . . .  | 36        |
| storage.set . . . . .          | 39        |
| storage.set.memory . . . . .   | 41        |
| storage.set.object . . . . .   | 43        |
| storage.unset . . . . .        | 45        |
| storage.unset.memory . . . . . | 47        |
| storage.unset.object . . . . . | 49        |
| <b>Index</b>                   | <b>52</b> |

---

|              |                      |
|--------------|----------------------|
| functionCall | <i>Function Call</i> |
|--------------|----------------------|

---

Description

For a given function and call, return a list of class 'functionCall' which can be hashed to provide a unique identifier for the function and parameters used for this call.

Usage

```
functionCall(f = sys.function(sys.parent()), call = sys.call(sys.parent()))
```

**Arguments**

|      |                                               |
|------|-----------------------------------------------|
| f    | function, defaults to the containing function |
| call | call, default to the containing call          |

**Value**

functionCall, a hashable form of the function call information

**Examples**

```
# my example function
my.function <- function (x, y) x+y

# create a new function call
my.functionCall <- functionCall(my.function, call("my.function", 10, 10))

# the function and arguments are now available
my.functionCall$f
my.functionCall$args

# using the default argument values to get the function call of the containing function
my.function2 <- function (x, y) functionCall()
my.functionCall2 <- my.function2(10, 10)

# the function and arguments are now available
my.functionCall2$f
my.functionCall2$args
```

---

hash

*Hash*

---

**Description**

Hashes a value into a string.

**Usage**

```
hash(value)
```

**Arguments**

|       |               |
|-------|---------------|
| value | value to hash |
|-------|---------------|

**Value**

hashed value as a string

## Examples

```
my.function <- function (x, y) x+y

# a list of values to hash
values <- list(
  "Hello world!",
  101,
  3.142,
  TRUE,
  my.function,
  (function (x, y) x+y),
  functionCall(my.function, call("my.function", 10, 10)),
  list(a=1, b=2, c="hello")
)

# hash the values in the list
(hashes <- lapply(values, hash))

# Note that functions with the same body will have the same hash
hashes[[5]] == hashes[[6]]
```

---

hash.default

*Hash*

---

## Description

Default hash function.

## Usage

```
## Default S3 method:
hash(value)
```

## Arguments

|       |               |
|-------|---------------|
| value | value to hash |
|-------|---------------|

## Value

hashed value as a string

## References

digest::digest(value)

**Examples**

```

my.function <- function (x, y) x+y

# a list of values to hash
values <- list(
  "Hello world!",
  101,
  3.142,
  TRUE,
  my.function,
  (function (x, y) x+y),
  functionCall(my.function, call("my.function", 10, 10)),
  list(a=1, b=2, c="hello")
)

# hash the values in the list
(hashes <- lapply(values, hash))

# Note that functions with the same body will have the same hash
hashes[[5]] == hashes[[6]]

```

---

|               |             |
|---------------|-------------|
| hash.function | <i>Hash</i> |
|---------------|-------------|

---

**Description**

Hashes a function, but considering the formals and body, thus the resulting hash is influenced by changes to signature and implementation.

**Usage**

```

## S3 method for class ``function``
hash(value)

```

**Arguments**

|       |               |
|-------|---------------|
| value | value to hash |
|-------|---------------|

**Value**

hashed value as a string

**Examples**

```

my.function <- function (x, y) x+y

# a list of values to hash
values <- list(
  "Hello world!",

```

```

    101,
    3.142,
    TRUE,
    my.function,
    (function (x, y) x+y),
    functionCall(my.function, call("my.function", 10, 10)),
    list(a=1, b=2, c="hello")
  )

# hash the values in the list
(hashes <- lapply(values, hash))

# Note that functions with the same body will have the same hash
hashes[[5]] == hashes[[6]]

```

---

|                   |             |
|-------------------|-------------|
| hash.functionCall | <i>Hash</i> |
|-------------------|-------------|

---

### Description

Hashes a function call, taking into account the values provided to the function call and unprovided default values. Ensures the order the parameters are provided does not change the outcome of the hash calculation.

### Usage

```
## S3 method for class 'functionCall'
hash(value)
```

### Arguments

value                      value to hash

### Value

hashed value as a string

### Examples

```

my.function <- function (x, y) x+y

# a list of values to hash
values <- list(
  "Hello world!",
  101,
  3.142,
  TRUE,
  my.function,
  (function (x, y) x+y),
  functionCall(my.function, call("my.function", 10, 10)),

```

```
list(a=1, b=2, c="hello")
)

# hash the values in the list
(hashes <- lapply(values, hash))

# Note that functions with the same body will have the same hash
hashes[[5]] == hashes[[6]]
```

---

hash.list

*Hash*

---

### Description

Hashes a list of items, generating a single unique hash value which is based on the combination of hashed list items.

### Usage

```
## S3 method for class 'list'
hash(value)
```

### Arguments

value                      value to hash

### Value

hashed value as a string

### Examples

```
my.function <- function (x, y) x+y

# a list of values to hash
values <- list(
  "Hello world!",
  101,
  3.142,
  TRUE,
  my.function,
  (function (x, y) x+y),
  functionCall(my.function, call("my.function", 10, 10)),
  list(a=1, b=2, c="hello")
)

# hash the values in the list
(hashes <- lapply(values, hash))

# Note that functions with the same body will have the same hash
hashes[[5]] == hashes[[6]]
```

---

 invalidate

*Invalidate Memo*


---

## Description

Invalidates cached values for a memoized function without affecting unrelated memo entries in shared storage.

Call without key to invalidate all cached values for that memo only. Pass key to invalidate a single cached call:

- a named/unnamed list of function arguments
- a single character cache key hash

## Usage

```
invalidate(f, key = NULL)
```

## Arguments

|     |                                                           |
|-----|-----------------------------------------------------------|
| f   | memo function                                             |
| key | optional cache key selector for single-entry invalidation |

## Value

Invisibly returns the memo function

## Examples

```
library(magrittr)

base.dir <- file.path(tempdir(), "memofunc-example-invalidate")
unlink(base.dir, recursive = TRUE, force = TRUE)

old.options <- options(memofunc.storage.provider = list(name = "file", base.dir = base.dir))

counter <- new.env(parent = emptyenv())
counter$executed <- 0

expensive <- function (value) {
  counter$executed <- counter$executed + 1
  value * 10
}

memo.expensive <- memo(expensive, id = "invalidate-example")

# warm cache for two keys
memo.expensive(1)
memo.expensive(2)
```

```
# invalidate a single entry by arguments
invalidate(memo.expensive, list(value = 1))

counter$executed <- 0
memo.expensive(1)
memo.expensive(2)
counter$executed

# invalidate all entries for this memo only
invalidate(memo.expensive)

counter$executed <- 0
memo.expensive(1)
memo.expensive(2)
counter$executed

options(old.options)
unlink(base.dir, recursive = TRUE, force = TRUE)
```

---

*is.memo**Is Memo*

---

## Description

Checks whether the passed function is a memo function.

## Usage

```
is.memo(f)
```

## Arguments

f                      function, memo or otherwise

## Value

TRUE if memo function, FALSE otherwise

---

*memo**Memo*

---

## Description

Creates a memoized function from a named or anonymous function. Calls to the memoized function are served from cache after the first execution for a given key.

Runtime controls:

- `memo.force = TRUE`: ignore an existing cached value and recompute
- `memo.dryrun = TRUE`: do not execute or cache; return TRUE if execution would occur, FALSE if a cache hit exists

By default, NULL results are not cached. Set `allow.null = TRUE` to cache NULL values as valid outcomes.

Hashing strategy for stored values:

- **id component**: explicit id when provided, otherwise an inferred function name when available
- **function component**: hash of the function formals and body (or `function_hash_override` when supplied)
- **call component**: hash of normalized call arguments (including defaulted arguments, ordered by argument name)

The final storage key is a hash over these three components. This means that, by default, changing a function's implementation invalidates old cache entries for future reads while leaving historical values in storage.

Matching keys only share cached values when memo calls use the same underlying storage. Separate in-memory memo instances do not share values, even when keys are identical.

## Usage

```
memo(f, id = NULL, function_hash_override = NULL, allow.null = FALSE)
```

## Arguments

|                                     |                                                                                                                       |
|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| <code>f</code>                      | function to memoise                                                                                                   |
| <code>id</code>                     | optional identifier used to scope cache keys; defaults to the function name when available                            |
| <code>function_hash_override</code> | optional override value used in place of the default function identity hash; can be used with explicit or inferred id |
| <code>allow.null</code>             | if TRUE, the memoized function caches NULL results; FALSE by default                                                  |

## Value

the memoed function

**Examples**

```

library(magrittr)

# --- Basic memoization -----

# a simple function
simple.function <- function (value) {
  print("Executing!")
  value
}

# memoize and keep the new function separate
simple.function.memo <- memo(simple.function)

# id defaults to the function name when available
attr(simple.function.memo, "memo.id")

# memoize in place
simple.function %<>% memo()

# memoize anonymous functions
simple.function2 <- (function (value) value) %>% memo()

# use an explicit id for anonymous or generated functions
anonymous.memo <- memo(function (value) value, id = "example/anon")

# use an explicit id to keep cache stable across renames or wrappers
# start from the original non-memoized function to avoid wrapping a memo twice
renamed.function <- function (value) {
  print("Executing!")
  value
}
renamed.memo <- memo(renamed.function, id = "simple.function")

# first call executes
simple.function(10)

# second call with same inputs is cached
simple.function(10)

# different inputs execute once, then cache
simple.function(20)
simple.function(20)

# --- Runtime controls -----

# force recomputation even if cached
simple.function(10, memo.force = TRUE)

# dry-run does not execute or write; it reports whether execution would occur
simple.function(30, memo.dryrun = TRUE)  # TRUE (not cached yet)
simple.function(30)                      # executes and caches

```

```

simple.function(30, memo.dryrun = TRUE)  # FALSE (already cached)

# --- NULL handling -----

# by default NULL results are not cached
null.default <- memo(function (value) {
  print("Executing NULL function")
  NULL
})
null.default(1)
null.default(1)

# allow.null = TRUE caches NULL values too
null.cached <- memo(function (value) {
  print("Executing NULL function with allow.null")
  NULL
}, allow.null = TRUE)
null.cached(1)
null.cached(1)

# --- Function hash override -----

# use persistent storage to demonstrate cache reuse across separate memo instances
base.dir <- file.path(tempdir(), "memofunc-example-function-hash")
unlink(base.dir, recursive = TRUE, force = TRUE)

old.options <- options(memofunc.storage.provider = list(name = "file", base.dir = base.dir))

counter <- new.env(parent = emptyenv())
counter$executed <- 0

f.v1 <- function (value) {
  counter$executed <- counter$executed + 1
  value
}

f.v2 <- function (value) {
  counter$executed <- counter$executed + 1
  value + 1
}

# default behavior: changed body creates a new function hash component
memo(f.v1, id = "shared-id")(100)
counter$executed <- 0
memo(f.v2, id = "shared-id")(100)
counter$executed

# override: both implementations share function hash component intentionally
memo(f.v1, id = "shared-id", function_hash_override = "stable-token")(200)
counter$executed <- 0
memo(f.v2, id = "shared-id", function_hash_override = "stable-token")(200)
counter$executed

```

```
# this reuse only works because both memos share the same file-backed store above
# with separate in-memory memo instances, matching keys still would not share values

options(old.options)
unlink(base.dir, recursive = TRUE, force = TRUE)

# --- Performance example -----

slow.function <- (function (value) Sys.sleep(value)) %>% memo(allow.null = TRUE)
system.time(slow.function(2))
system.time(slow.function(2))
```

---

|            |                   |
|------------|-------------------|
| memo.cache | <i>Memo Cache</i> |
|------------|-------------------|

---

**Description**

Gets the cache associated with a memo function allowing further manipulation and control of the underlying values being stored.

Execution is stopped if function passed is not a valid memoed function.

**Usage**

```
memo.cache(f)
```

**Arguments**

f                      memo function

**Value**

Cache storing values for memoed function.

---

|               |                      |
|---------------|----------------------|
| memo.function | <i>Memo Function</i> |
|---------------|----------------------|

---

**Description**

Gets the original function that was memoized.

Execution is stopped if function passed is not a valid memoed function.

**Usage**

```
memo.function(f)
```

**Arguments**

f                      memo function

**Value**

Original unmemoized function.

---

|               |                           |
|---------------|---------------------------|
| storage.clear | <i>Clear the storage.</i> |
|---------------|---------------------------|

---

**Description**

Clear the given storage of all keys and their values.

**Usage**

```
storage.clear(storage)
```

**Arguments**

storage                initialized storage

**Value**

Invisibly returns storage

**Examples**

```
library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
```

```

sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")

# initialize file storage (direct backend; simplest local-only path)
base.dir <- file.path(tempdir(), "memofunc-storage")
file.storage <- storage.init("file", base.dir = base.dir)

# set and retrieve a value from file storage (direct backend)
storage.set(file.storage, "name", "Roy Wetherall")
storage.get(file.storage, "name")

# initialize object storage using a provider name (provider-backed interface)
# this uses the same local file backend now, but lets you swap to azure.blob/s3 later
object.storage <- storage.init("object", provider = "file", base.dir = base.dir)
storage.set(object.storage, "name", "Roy Wetherall")
storage.get(object.storage, "name")

# set a default provider via options
old.options <- options(memofunc.storage.provider = list(name = "file", base.dir = base.dir))
on.exit(options(old.options), add = TRUE)
default.storage <- storage.init()
storage.get(default.storage, "name")

# Azure Blob provider example (requires AzureStor + credentials)
if (requireNamespace("AzureStor", quietly = TRUE)) {
  account <- Sys.getenv("AZURE_STORAGE_ACCOUNT")
  container <- Sys.getenv("AZURE_STORAGE_CONTAINER")
  key <- Sys.getenv("AZURE_STORAGE_KEY")
  token <- Sys.getenv("AZURE_STORAGE_TOKEN")

  if (account != "" && container != "" && (key != "" || token != "")) {
    azure.storage <- storage.init(
      "object",
      provider = "azure.blob",
      account = account,
      container = container,
      key = if (key == "") NULL else key,
      token = if (token == "") NULL else token,
      prefix = "memofunc-example"
  )
  }
}

```

```

    )

    storage.set(azure.storage, "name", "Roy Wetherall")
    storage.get(azure.storage, "name")
  }
}

```

---

storage.clear.memory    *Clear the memory store.*

---

## Description

Clear the given storage of all keys and their values.

## Usage

```
## S3 method for class 'memory'
storage.clear(storage)
```

## Arguments

storage                  initialized storage

## Value

Invisibly returns storage

## Examples

```

library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage

```

```

sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")

# initialize file storage (direct backend; simplest local-only path)
base.dir <- file.path(tempdir(), "memofunc-storage")
file.storage <- storage.init("file", base.dir = base.dir)

# set and retrieve a value from file storage (direct backend)
storage.set(file.storage, "name", "Roy Wetherall")
storage.get(file.storage, "name")

# initialize object storage using a provider name (provider-backed interface)
# this uses the same local file backend now, but lets you swap to azure.blob/s3 later
object.storage <- storage.init("object", provider = "file", base.dir = base.dir)
storage.set(object.storage, "name", "Roy Wetherall")
storage.get(object.storage, "name")

# set a default provider via options
old.options <- options(memofunc.storage.provider = list(name = "file", base.dir = base.dir))
on.exit(options(old.options), add = TRUE)
default.storage <- storage.init()
storage.get(default.storage, "name")

# Azure Blob provider example (requires AzureStor + credentials)
if (requireNamespace("AzureStor", quietly = TRUE)) {
  account <- Sys.getenv("AZURE_STORAGE_ACCOUNT")
  container <- Sys.getenv("AZURE_STORAGE_CONTAINER")
  key <- Sys.getenv("AZURE_STORAGE_KEY")
  token <- Sys.getenv("AZURE_STORAGE_TOKEN")

  if (account != "" && container != "" && (key != "" || token != "")) {
    azure.storage <- storage.init(
      "object",
      provider = "azure.blob",
      account = account,
      container = container,
      key = if (key == "") NULL else key,
      token = if (token == "") NULL else token,
      prefix = "memofunc-example"
  )
  }
}

```

```

    )

    storage.set(azure.storage, "name", "Roy Wetherall")
    storage.get(azure.storage, "name")
  }
}

```

---

storage.clear.object    *Clear the object store.*

---

## Description

Clear the given storage of all keys and their values.

## Usage

```
## S3 method for class 'object'
storage.clear(storage)
```

## Arguments

storage                  initialized storage

## Value

Invisibly returns storage

## Examples

```

library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage

```

```

sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")

# initialize file storage (direct backend; simplest local-only path)
base.dir <- file.path(tempdir(), "memofunc-storage")
file.storage <- storage.init("file", base.dir = base.dir)

# set and retrieve a value from file storage (direct backend)
storage.set(file.storage, "name", "Roy Wetherall")
storage.get(file.storage, "name")

# initialize object storage using a provider name (provider-backed interface)
# this uses the same local file backend now, but lets you swap to azure.blob/s3 later
object.storage <- storage.init("object", provider = "file", base.dir = base.dir)
storage.set(object.storage, "name", "Roy Wetherall")
storage.get(object.storage, "name")

# set a default provider via options
old.options <- options(memofunc.storage.provider = list(name = "file", base.dir = base.dir))
on.exit(options(old.options), add = TRUE)
default.storage <- storage.init()
storage.get(default.storage, "name")

# Azure Blob provider example (requires AzureStor + credentials)
if (requireNamespace("AzureStor", quietly = TRUE)) {
  account <- Sys.getenv("AZURE_STORAGE_ACCOUNT")
  container <- Sys.getenv("AZURE_STORAGE_CONTAINER")
  key <- Sys.getenv("AZURE_STORAGE_KEY")
  token <- Sys.getenv("AZURE_STORAGE_TOKEN")

  if (account != "" && container != "" && (key != "" || token != "")) {
    azure.storage <- storage.init(
      "object",
      provider = "azure.blob",
      account = account,
      container = container,
      key = if (key == "") NULL else key,
      token = if (token == "") NULL else token,
      prefix = "memofunc-example"
  )
  }
}

```

```

    )

    storage.set(azure.storage, "name", "Roy Wetherall")
    storage.get(azure.storage, "name")
  }
}

```

---

storage.get

*Get value from a store.*

---

### Description

Gets a value, for a given key, from the store.

If there is no corresponding value for the key, then NULL is returned.

### Usage

```
storage.get(storage, key)
```

### Arguments

|         |                           |
|---------|---------------------------|
| storage | initialized storage       |
| key     | key to retrieve value for |

### Value

Stored value for the key, NULL otherwise.

### Examples

```

library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

```

```

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")

# initialize file storage (direct backend; simplest local-only path)
base.dir <- file.path(tempdir(), "memofunc-storage")
file.storage <- storage.init("file", base.dir = base.dir)

# set and retrieve a value from file storage (direct backend)
storage.set(file.storage, "name", "Roy Wetherall")
storage.get(file.storage, "name")

# initialize object storage using a provider name (provider-backed interface)
# this uses the same local file backend now, but lets you swap to azure.blob/s3 later
object.storage <- storage.init("object", provider = "file", base.dir = base.dir)
storage.set(object.storage, "name", "Roy Wetherall")
storage.get(object.storage, "name")

# set a default provider via options
old.options <- options(memofunc.storage.provider = list(name = "file", base.dir = base.dir))
on.exit(options(old.options), add = TRUE)
default.storage <- storage.init()
storage.get(default.storage, "name")

# Azure Blob provider example (requires AzureStor + credentials)
if (requireNamespace("AzureStor", quietly = TRUE)) {
  account <- Sys.getenv("AZURE_STORAGE_ACCOUNT")
  container <- Sys.getenv("AZURE_STORAGE_CONTAINER")
  key <- Sys.getenv("AZURE_STORAGE_KEY")
  token <- Sys.getenv("AZURE_STORAGE_TOKEN")

  if (account != "" && container != "" && (key != "" || token != "")) {
    azure.storage <- storage.init(
      "object",
      provider = "azure.blob",
      account = account,
      container = container,
      key = if (key == "") NULL else key,

```

```

        token = if (token == "") NULL else token,
        prefix = "memofunc-example"
    )

    storage.set(azure.storage, "name", "Roy Wetherall")
    storage.get(azure.storage, "name")
  }
}

```

---

|                    |                                         |
|--------------------|-----------------------------------------|
| storage.get.memory | <i>Get a value from a memory store.</i> |
|--------------------|-----------------------------------------|

---

## Description

Gets a value, for a given key, from the store.

If there is no corresponding value for the key, then NULL is returned.

## Usage

```

## S3 method for class 'memory'
storage.get(storage, key)

```

## Arguments

|         |                           |
|---------|---------------------------|
| storage | initialized storage       |
| key     | key to retrieve value for |

## Value

Stored value for the key, NULL otherwise.

## Examples

```

library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

```

```

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")

# initialize file storage (direct backend; simplest local-only path)
base.dir <- file.path(tempdir(), "memofunc-storage")
file.storage <- storage.init("file", base.dir = base.dir)

# set and retrieve a value from file storage (direct backend)
storage.set(file.storage, "name", "Roy Wetherall")
storage.get(file.storage, "name")

# initialize object storage using a provider name (provider-backed interface)
# this uses the same local file backend now, but lets you swap to azure.blob/s3 later
object.storage <- storage.init("object", provider = "file", base.dir = base.dir)
storage.set(object.storage, "name", "Roy Wetherall")
storage.get(object.storage, "name")

# set a default provider via options
old.options <- options(memofunc.storage.provider = list(name = "file", base.dir = base.dir))
on.exit(options(old.options), add = TRUE)
default.storage <- storage.init()
storage.get(default.storage, "name")

# Azure Blob provider example (requires AzureStor + credentials)
if (requireNamespace("AzureStor", quietly = TRUE)) {
  account <- Sys.getenv("AZURE_STORAGE_ACCOUNT")
  container <- Sys.getenv("AZURE_STORAGE_CONTAINER")
  key <- Sys.getenv("AZURE_STORAGE_KEY")
  token <- Sys.getenv("AZURE_STORAGE_TOKEN")

  if (account != "" && container != "" && (key != "" || token != "")) {
    azure.storage <- storage.init(
      "object",
      provider = "azure.blob",

```

```

        account = account,
        container = container,
        key = if (key == "") NULL else key,
        token = if (token == "") NULL else token,
        prefix = "memofunc-example"
    )

    storage.set(azure.storage, "name", "Roy Wetherall")
    storage.get(azure.storage, "name")
  }
}

```

---

|                    |                                          |
|--------------------|------------------------------------------|
| storage.get.object | <i>Get a value from an object store.</i> |
|--------------------|------------------------------------------|

---

### Description

Gets a value, for a given key, from the store.

If there is no corresponding value for the key, then NULL is returned.

### Usage

```

## S3 method for class 'object'
storage.get(storage, key)

```

### Arguments

|         |                           |
|---------|---------------------------|
| storage | initialized storage       |
| key     | key to retrieve value for |

### Value

Stored value for the key, NULL otherwise.

### Examples

```

library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

```

```

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")

# initialize file storage (direct backend; simplest local-only path)
base.dir <- file.path(tempdir(), "memofunc-storage")
file.storage <- storage.init("file", base.dir = base.dir)

# set and retrieve a value from file storage (direct backend)
storage.set(file.storage, "name", "Roy Wetherall")
storage.get(file.storage, "name")

# initialize object storage using a provider name (provider-backed interface)
# this uses the same local file backend now, but lets you swap to azure.blob/s3 later
object.storage <- storage.init("object", provider = "file", base.dir = base.dir)
storage.set(object.storage, "name", "Roy Wetherall")
storage.get(object.storage, "name")

# set a default provider via options
old.options <- options(memofunc.storage.provider = list(name = "file", base.dir = base.dir))
on.exit(options(old.options), add = TRUE)
default.storage <- storage.init()
storage.get(default.storage, "name")

# Azure Blob provider example (requires AzureStor + credentials)
if (requireNamespace("AzureStor", quietly = TRUE)) {
  account <- Sys.getenv("AZURE_STORAGE_ACCOUNT")
  container <- Sys.getenv("AZURE_STORAGE_CONTAINER")
  key <- Sys.getenv("AZURE_STORAGE_KEY")
  token <- Sys.getenv("AZURE_STORAGE_TOKEN")

  if (account != "" && container != "" && (key != "" || token != "")) {

```

```

    azure.storage <- storage.init(
      "object",
      provider = "azure.blob",
      account = account,
      container = container,
      key = if (key == "") NULL else key,
      token = if (token == "") NULL else token,
      prefix = "memofunc-example"
    )

    storage.set(azure.storage, "name", "Roy Wetherall")
    storage.get(azure.storage, "name")
  }
}

```

---

storage.has

*Has key has been used to store a value?*


---

### Description

Indicates if a given key has a associated value stored in the storage or not.

### Usage

```
storage.has(storage, key)
```

### Arguments

|         |                               |
|---------|-------------------------------|
| storage | initialized storage           |
| key     | key to check for stored value |

### Value

TRUE if key has an associated stored value, FALSE otherwise.

### Examples

```

library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%

```

```

    storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")

# initialize file storage (direct backend; simplest local-only path)
base.dir <- file.path(tempdir(), "memofunc-storage")
file.storage <- storage.init("file", base.dir = base.dir)

# set and retrieve a value from file storage (direct backend)
storage.set(file.storage, "name", "Roy Wetherall")
storage.get(file.storage, "name")

# initialize object storage using a provider name (provider-backed interface)
# this uses the same local file backend now, but lets you swap to azure.blob/s3 later
object.storage <- storage.init("object", provider = "file", base.dir = base.dir)
storage.set(object.storage, "name", "Roy Wetherall")
storage.get(object.storage, "name")

# set a default provider via options
old.options <- options(memofunc.storage.provider = list(name = "file", base.dir = base.dir))
on.exit(options(old.options), add = TRUE)
default.storage <- storage.init()
storage.get(default.storage, "name")

# Azure Blob provider example (requires AzureStor + credentials)
if (requireNamespace("AzureStor", quietly = TRUE)) {
  account <- Sys.getenv("AZURE_STORAGE_ACCOUNT")
  container <- Sys.getenv("AZURE_STORAGE_CONTAINER")
  key <- Sys.getenv("AZURE_STORAGE_KEY")
  token <- Sys.getenv("AZURE_STORAGE_TOKEN")

```

```

if (account != "" && container != "" && (key != "" || token != "")) {
  azure.storage <- storage.init(
    "object",
    provider = "azure.blob",
    account = account,
    container = container,
    key = if (key == "") NULL else key,
    token = if (token == "") NULL else token,
    prefix = "memofunc-example"
  )

  storage.set(azure.storage, "name", "Roy Wetherall")
  storage.get(azure.storage, "name")
}
}

```

---

|                    |                                                                  |
|--------------------|------------------------------------------------------------------|
| storage.has.memory | <i>Has key has been used to store a value in a memory store?</i> |
|--------------------|------------------------------------------------------------------|

---

## Description

Indicates if a given key has a associated value stored in the storage or not.

## Usage

```

## S3 method for class 'memory'
storage.has(storage, key)

```

## Arguments

|         |                               |
|---------|-------------------------------|
| storage | initialized storage           |
| key     | key to check for stored value |

## Value

TRUE if key has an associated stored value, FALSE otherwise.

## Examples

```

library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%

```

```

storage.set("age", 45) %>%
storage.set("alive", TRUE) %>%
storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")

# initialize file storage (direct backend; simplest local-only path)
base.dir <- file.path(tempdir(), "memofunc-storage")
file.storage <- storage.init("file", base.dir = base.dir)

# set and retrieve a value from file storage (direct backend)
storage.set(file.storage, "name", "Roy Wetherall")
storage.get(file.storage, "name")

# initialize object storage using a provider name (provider-backed interface)
# this uses the same local file backend now, but lets you swap to azure.blob/s3 later
object.storage <- storage.init("object", provider = "file", base.dir = base.dir)
storage.set(object.storage, "name", "Roy Wetherall")
storage.get(object.storage, "name")

# set a default provider via options
old.options <- options(memofunc.storage.provider = list(name = "file", base.dir = base.dir))
on.exit(options(old.options), add = TRUE)
default.storage <- storage.init()
storage.get(default.storage, "name")

# Azure Blob provider example (requires AzureStor + credentials)
if (requireNamespace("AzureStor", quietly = TRUE)) {
  account <- Sys.getenv("AZURE_STORAGE_ACCOUNT")
  container <- Sys.getenv("AZURE_STORAGE_CONTAINER")
  key <- Sys.getenv("AZURE_STORAGE_KEY")

```

```

token <- Sys.getenv("AZURE_STORAGE_TOKEN")

if (account != "" && container != "" && (key != "" || token != "")) {
  azure.storage <- storage.init(
    "object",
    provider = "azure.blob",
    account = account,
    container = container,
    key = if (key == "") NULL else key,
    token = if (token == "") NULL else token,
    prefix = "memofunc-example"
  )

  storage.set(azure.storage, "name", "Roy Wetherall")
  storage.get(azure.storage, "name")
}
}

```

---

|                    |                                                                   |
|--------------------|-------------------------------------------------------------------|
| storage.has.object | <i>Has key has been used to store a value in an object store?</i> |
|--------------------|-------------------------------------------------------------------|

---

### Description

Indicates if a given key has a associated value stored in the storage or not.

### Usage

```

## S3 method for class 'object'
storage.has(storage, key)

```

### Arguments

|         |                               |
|---------|-------------------------------|
| storage | initialized storage           |
| key     | key to check for stored value |

### Value

TRUE if key has an associated stored value, FALSE otherwise.

### Examples

```

library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

```

```

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")

# initialize file storage (direct backend; simplest local-only path)
base.dir <- file.path(tempdir(), "memofunc-storage")
file.storage <- storage.init("file", base.dir = base.dir)

# set and retrieve a value from file storage (direct backend)
storage.set(file.storage, "name", "Roy Wetherall")
storage.get(file.storage, "name")

# initialize object storage using a provider name (provider-backed interface)
# this uses the same local file backend now, but lets you swap to azure.blob/s3 later
object.storage <- storage.init("object", provider = "file", base.dir = base.dir)
storage.set(object.storage, "name", "Roy Wetherall")
storage.get(object.storage, "name")

# set a default provider via options
old.options <- options(memofunc.storage.provider = list(name = "file", base.dir = base.dir))
on.exit(options(old.options), add = TRUE)
default.storage <- storage.init()
storage.get(default.storage, "name")

# Azure Blob provider example (requires AzureStor + credentials)
if (requireNamespace("AzureStor", quietly = TRUE)) {
  account <- Sys.getenv("AZURE_STORAGE_ACCOUNT")

```

```

container <- Sys.getenv("AZURE_STORAGE_CONTAINER")
key <- Sys.getenv("AZURE_STORAGE_KEY")
token <- Sys.getenv("AZURE_STORAGE_TOKEN")

if (account != "" && container != "" && (key != "" || token != "")) {
  azure.storage <- storage.init(
    "object",
    provider = "azure.blob",
    account = account,
    container = container,
    key = if (key == "") NULL else key,
    token = if (token == "") NULL else token,
    prefix = "memofunc-example"
  )

  storage.set(azure.storage, "name", "Roy Wetherall")
  storage.get(azure.storage, "name")
}
}

```

---

storage.init

*Initialize a store.*


---

## Description

Initlaize storage for name value pairs based on provided type.

Available types of storage include:

- memory: transient in-memory storage
- file: persistent storage, using local file storage
- object: provider-backed object storage

Use file when you want a simple local file store without provider configuration. Use object with a provider (e.g. file, azure.blob) when you want a consistent, pluggable interface across storage backends. This allows the same code path to swap between local and cloud providers.

Additional paramters may be provided when initializing different types of storage. If storage.type is not provided and memofunc.storage.provider is set, then the provider is used to initialize storage.

Providers are resolved by name (for example file or azure.blob), with synonyms such as local or azure mapping to their canonical names. The Azure provider requires the AzureStor package.

See specific storage types for details.

## Usage

```

storage.init(
  storage.type = storage.memory.class,
  provider = getOption("memofunc.storage.provider", NULL),
  ...
)

```

**Arguments**

|              |                                                                 |
|--------------|-----------------------------------------------------------------|
| storage.type | storage type to initialize, defaults to memory                  |
| provider     | optional provider name or provider configuration                |
| ...          | additional configuration values used by storage implementations |

**Value**

List containing characteristics particular to the storage implementation, including:

- type: the storage type field

**Examples**

```
library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")
```

```

# initialize file storage (direct backend; simplest local-only path)
base.dir <- file.path(tempdir(), "memofunc-storage")
file.storage <- storage.init("file", base.dir = base.dir)

# set and retrieve a value from file storage (direct backend)
storage.set(file.storage, "name", "Roy Wetherall")
storage.get(file.storage, "name")

# initialize object storage using a provider name (provider-backed interface)
# this uses the same local file backend now, but lets you swap to azure.blob/s3 later
object.storage <- storage.init("object", provider = "file", base.dir = base.dir)
storage.set(object.storage, "name", "Roy Wetherall")
storage.get(object.storage, "name")

# set a default provider via options
old.options <- options(memofunc.storage.provider = list(name = "file", base.dir = base.dir))
on.exit(options(old.options), add = TRUE)
default.storage <- storage.init()
storage.get(default.storage, "name")

# Azure Blob provider example (requires AzureStor + credentials)
if (requireNamespace("AzureStor", quietly = TRUE)) {
  account <- Sys.getenv("AZURE_STORAGE_ACCOUNT")
  container <- Sys.getenv("AZURE_STORAGE_CONTAINER")
  key <- Sys.getenv("AZURE_STORAGE_KEY")
  token <- Sys.getenv("AZURE_STORAGE_TOKEN")

  if (account != "" && container != "" && (key != "" || token != "")) {
    azure.storage <- storage.init(
      "object",
      provider = "azure.blob",
      account = account,
      container = container,
      key = if (key == "") NULL else key,
      token = if (token == "") NULL else token,
      prefix = "memofunc-example"
    )

    storage.set(azure.storage, "name", "Roy Wetherall")
    storage.get(azure.storage, "name")
  }
}

```

---

storage.init.memory     *Initialize a memory store.*

---

## Description

Initlaize memory storage, used to hold and retrieve values in memory.

The storage type is expected to specified as memory.

This storage is transient.

**Usage**

```
## S3 method for class 'memory'
storage.init(storage.type = storage.memory.class, ...)
```

**Arguments**

```
storage.type    storage type to initialize, defaults to memory
...             additional configuration values used by storage implementations
```

**Value**

List containing characteristics particular to the storage implementation, including:

- \$type - the storage type

**Examples**

```
library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)
```

```

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")

# initialize file storage (direct backend; simplest local-only path)
base.dir <- file.path(tempdir(), "memofunc-storage")
file.storage <- storage.init("file", base.dir = base.dir)

# set and retrieve a value from file storage (direct backend)
storage.set(file.storage, "name", "Roy Wetherall")
storage.get(file.storage, "name")

# initialize object storage using a provider name (provider-backed interface)
# this uses the same local file backend now, but lets you swap to azure.blob/s3 later
object.storage <- storage.init("object", provider = "file", base.dir = base.dir)
storage.set(object.storage, "name", "Roy Wetherall")
storage.get(object.storage, "name")

# set a default provider via options
old.options <- options(memofunc.storage.provider = list(name = "file", base.dir = base.dir))
on.exit(options(old.options), add = TRUE)
default.storage <- storage.init()
storage.get(default.storage, "name")

# Azure Blob provider example (requires AzureStor + credentials)
if (requireNamespace("AzureStor", quietly = TRUE)) {
  account <- Sys.getenv("AZURE_STORAGE_ACCOUNT")
  container <- Sys.getenv("AZURE_STORAGE_CONTAINER")
  key <- Sys.getenv("AZURE_STORAGE_KEY")
  token <- Sys.getenv("AZURE_STORAGE_TOKEN")

  if (account != "" && container != "" && (key != "" || token != "")) {
    azure.storage <- storage.init(
      "object",
      provider = "azure.blob",
      account = account,
      container = container,
      key = if (key == "") NULL else key,
      token = if (token == "") NULL else token,
      prefix = "memofunc-example"
    )

    storage.set(azure.storage, "name", "Roy Wetherall")
    storage.get(azure.storage, "name")
  }
}

```

---

|                     |                                    |
|---------------------|------------------------------------|
| storage.init.object | <i>Initialize an object store.</i> |
|---------------------|------------------------------------|

---

**Description**

Initializes object storage backed by a provider.

The provider must supply methods for putting, getting, checking, deleting, and clearing stored values.

**Usage**

```
## S3 method for class 'object'
storage.init(storage.type = storage.object.class, provider, ...)
```

**Arguments**

|              |                                                                 |
|--------------|-----------------------------------------------------------------|
| storage.type | storage type to initialize, defaults to memory                  |
| provider     | provider implementation exposing put/get/exists/delete/clear    |
| ...          | additional configuration values used by storage implementations |

**Value**

List containing characteristics particular to the storage implementation, including:

- \$type - the storage type

**Examples**

```
library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))
```

```

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")

# initialize file storage (direct backend; simplest local-only path)
base.dir <- file.path(tempdir(), "memofunc-storage")
file.storage <- storage.init("file", base.dir = base.dir)

# set and retrieve a value from file storage (direct backend)
storage.set(file.storage, "name", "Roy Wetherall")
storage.get(file.storage, "name")

# initialize object storage using a provider name (provider-backed interface)
# this uses the same local file backend now, but lets you swap to azure.blob/s3 later
object.storage <- storage.init("object", provider = "file", base.dir = base.dir)
storage.set(object.storage, "name", "Roy Wetherall")
storage.get(object.storage, "name")

# set a default provider via options
old.options <- options(memofunc.storage.provider = list(name = "file", base.dir = base.dir))
on.exit(options(old.options), add = TRUE)
default.storage <- storage.init()
storage.get(default.storage, "name")

# Azure Blob provider example (requires AzureStor + credentials)
if (requireNamespace("AzureStor", quietly = TRUE)) {
  account <- Sys.getenv("AZURE_STORAGE_ACCOUNT")
  container <- Sys.getenv("AZURE_STORAGE_CONTAINER")
  key <- Sys.getenv("AZURE_STORAGE_KEY")
  token <- Sys.getenv("AZURE_STORAGE_TOKEN")

  if (account != "" && container != "" && (key != "" || token != "")) {
    azure.storage <- storage.init(
      "object",
      provider = "azure.blob",
      account = account,
      container = container,
      key = if (key == "") NULL else key,
      token = if (token == "") NULL else token,
      prefix = "memofunc-example"
    )

    storage.set(azure.storage, "name", "Roy Wetherall")
    storage.get(azure.storage, "name")
  }
}

```

```
}
```

---

|             |                                |
|-------------|--------------------------------|
| storage.set | <i>Set value into a store.</i> |
|-------------|--------------------------------|

---

## Description

Stores a value for a given key.

If there is already a value stored for the key provided, then the existing value is overridden with the new value.

## Usage

```
storage.set(storage, key, value)
```

## Arguments

|         |                            |
|---------|----------------------------|
| storage | initialized storage        |
| key     | key to store value against |
| value   | value to store             |

## Value

Invisibly returns storage

## Examples

```
library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
```

```

sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")

# initialize file storage (direct backend; simplest local-only path)
base.dir <- file.path(tempdir(), "memofunc-storage")
file.storage <- storage.init("file", base.dir = base.dir)

# set and retrieve a value from file storage (direct backend)
storage.set(file.storage, "name", "Roy Wetherall")
storage.get(file.storage, "name")

# initialize object storage using a provider name (provider-backed interface)
# this uses the same local file backend now, but lets you swap to azure.blob/s3 later
object.storage <- storage.init("object", provider = "file", base.dir = base.dir)
storage.set(object.storage, "name", "Roy Wetherall")
storage.get(object.storage, "name")

# set a default provider via options
old.options <- options(memofunc.storage.provider = list(name = "file", base.dir = base.dir))
on.exit(options(old.options), add = TRUE)
default.storage <- storage.init()
storage.get(default.storage, "name")

# Azure Blob provider example (requires AzureStor + credentials)
if (requireNamespace("AzureStor", quietly = TRUE)) {
  account <- Sys.getenv("AZURE_STORAGE_ACCOUNT")
  container <- Sys.getenv("AZURE_STORAGE_CONTAINER")
  key <- Sys.getenv("AZURE_STORAGE_KEY")
  token <- Sys.getenv("AZURE_STORAGE_TOKEN")

  if (account != "" && container != "" && (key != "" || token != "")) {
    azure.storage <- storage.init(
      "object",
      provider = "azure.blob",
      account = account,
      container = container,
      key = if (key == "") NULL else key,
      token = if (token == "") NULL else token,
      prefix = "memofunc-example"
  )
  }
}

```

```

    )

    storage.set(azure.storage, "name", "Roy Wetherall")
    storage.get(azure.storage, "name")
  }
}

```

---

|                    |                                       |
|--------------------|---------------------------------------|
| storage.set.memory | <i>Set value into a memory store.</i> |
|--------------------|---------------------------------------|

---

## Description

Stores a value for a given key.

If there is already a value stored for the key provided, then the existing value is overridden with the new value.

## Usage

```

## S3 method for class 'memory'
storage.set(storage, key, value)

```

## Arguments

|         |                            |
|---------|----------------------------|
| storage | initialized storage        |
| key     | key to store value against |
| value   | value to store             |

## Value

Invisibly returns storage

## Examples

```

library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set

```

```

if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")

# initialize file storage (direct backend; simplest local-only path)
base.dir <- file.path(tempdir(), "memofunc-storage")
file.storage <- storage.init("file", base.dir = base.dir)

# set and retrieve a value from file storage (direct backend)
storage.set(file.storage, "name", "Roy Wetherall")
storage.get(file.storage, "name")

# initialize object storage using a provider name (provider-backed interface)
# this uses the same local file backend now, but lets you swap to azure.blob/s3 later
object.storage <- storage.init("object", provider = "file", base.dir = base.dir)
storage.set(object.storage, "name", "Roy Wetherall")
storage.get(object.storage, "name")

# set a default provider via options
old.options <- options(memofunc.storage.provider = list(name = "file", base.dir = base.dir))
on.exit(options(old.options), add = TRUE)
default.storage <- storage.init()
storage.get(default.storage, "name")

# Azure Blob provider example (requires AzureStor + credentials)
if (requireNamespace("AzureStor", quietly = TRUE)) {
  account <- Sys.getenv("AZURE_STORAGE_ACCOUNT")
  container <- Sys.getenv("AZURE_STORAGE_CONTAINER")
  key <- Sys.getenv("AZURE_STORAGE_KEY")
  token <- Sys.getenv("AZURE_STORAGE_TOKEN")

  if (account != "" && container != "" && (key != "" || token != "")) {
    azure.storage <- storage.init(
      "object",

```

```

        provider = "azure.blob",
        account = account,
        container = container,
        key = if (key == "") NULL else key,
        token = if (token == "") NULL else token,
        prefix = "memofunc-example"
    )

    storage.set(azure.storage, "name", "Roy Wetherall")
    storage.get(azure.storage, "name")
  }
}

```

---

|                    |                                        |
|--------------------|----------------------------------------|
| storage.set.object | <i>Set value into an object store.</i> |
|--------------------|----------------------------------------|

---

### Description

Stores a value for a given key.

If there is already a value stored for the key provided, then the existing value is overridden with the new value.

### Usage

```

## S3 method for class 'object'
storage.set(storage, key, value)

```

### Arguments

|         |                            |
|---------|----------------------------|
| storage | initialized storage        |
| key     | key to store value against |
| value   | value to store             |

### Value

Invisibly returns storage

### Examples

```

library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more

```

```

my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")

# initialize file storage (direct backend; simplest local-only path)
base.dir <- file.path(tempdir(), "memofunc-storage")
file.storage <- storage.init("file", base.dir = base.dir)

# set and retrieve a value from file storage (direct backend)
storage.set(file.storage, "name", "Roy Wetherall")
storage.get(file.storage, "name")

# initialize object storage using a provider name (provider-backed interface)
# this uses the same local file backend now, but lets you swap to azure.blob/s3 later
object.storage <- storage.init("object", provider = "file", base.dir = base.dir)
storage.set(object.storage, "name", "Roy Wetherall")
storage.get(object.storage, "name")

# set a default provider via options
old.options <- options(memofunc.storage.provider = list(name = "file", base.dir = base.dir))
on.exit(options(old.options), add = TRUE)
default.storage <- storage.init()
storage.get(default.storage, "name")

# Azure Blob provider example (requires AzureStor + credentials)
if (requireNamespace("AzureStor", quietly = TRUE)) {
  account <- Sys.getenv("AZURE_STORAGE_ACCOUNT")
  container <- Sys.getenv("AZURE_STORAGE_CONTAINER")

```

```

key <- Sys.getenv("AZURE_STORAGE_KEY")
token <- Sys.getenv("AZURE_STORAGE_TOKEN")

if (account != "" && container != "" && (key != "" || token != "")) {
  azure.storage <- storage.init(
    "object",
    provider = "azure.blob",
    account = account,
    container = container,
    key = if (key == "") NULL else key,
    token = if (token == "") NULL else token,
    prefix = "memofunc-example"
  )

  storage.set(azure.storage, "name", "Roy Wetherall")
  storage.get(azure.storage, "name")
}
}

```

---

storage.unset

*Unset a value that corresponds to a key within a store.*


---

### Description

Unsets the value stored for a given key.

If there is no value for the key provided no action is taken.

### Usage

```
storage.unset(storage, key)
```

### Arguments

|         |                                |
|---------|--------------------------------|
| storage | initialized storage            |
| key     | key whose value is to be unset |

### Value

Invisibly returns storage

### Examples

```

library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

```

```

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")

# initialize file storage (direct backend; simplest local-only path)
base.dir <- file.path(tempdir(), "memofunc-storage")
file.storage <- storage.init("file", base.dir = base.dir)

# set and retrieve a value from file storage (direct backend)
storage.set(file.storage, "name", "Roy Wetherall")
storage.get(file.storage, "name")

# initialize object storage using a provider name (provider-backed interface)
# this uses the same local file backend now, but lets you swap to azure.blob/s3 later
object.storage <- storage.init("object", provider = "file", base.dir = base.dir)
storage.set(object.storage, "name", "Roy Wetherall")
storage.get(object.storage, "name")

# set a default provider via options
old.options <- options(memofunc.storage.provider = list(name = "file", base.dir = base.dir))
on.exit(options(old.options), add = TRUE)
default.storage <- storage.init()
storage.get(default.storage, "name")

# Azure Blob provider example (requires AzureStor + credentials)
if (requireNamespace("AzureStor", quietly = TRUE)) {

```

```

account <- Sys.getenv("AZURE_STORAGE_ACCOUNT")
container <- Sys.getenv("AZURE_STORAGE_CONTAINER")
key <- Sys.getenv("AZURE_STORAGE_KEY")
token <- Sys.getenv("AZURE_STORAGE_TOKEN")

if (account != "" && container != "" && (key != "" || token != "")) {
  azure.storage <- storage.init(
    "object",
    provider = "azure.blob",
    account = account,
    container = container,
    key = if (key == "") NULL else key,
    token = if (token == "") NULL else token,
    prefix = "memofunc-example"
  )

  storage.set(azure.storage, "name", "Roy Wetherall")
  storage.get(azure.storage, "name")
}
}

```

---

storage.unset.memory    *Unset a value that corresponds to a key within a memory store.*

---

## Description

Unsets the value stored for a given key.

If there is no value for the key provided no action is taken.

## Usage

```

## S3 method for class 'memory'
storage.unset(storage, key)

```

## Arguments

|         |                                |
|---------|--------------------------------|
| storage | initialized storage            |
| key     | key whose value is to be unset |

## Value

Invisibly returns storage

**Examples**

```

library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")

# initialize file storage (direct backend; simplest local-only path)
base.dir <- file.path(tempdir(), "memofunc-storage")
file.storage <- storage.init("file", base.dir = base.dir)

# set and retrieve a value from file storage (direct backend)
storage.set(file.storage, "name", "Roy Wetherall")
storage.get(file.storage, "name")

# initialize object storage using a provider name (provider-backed interface)
# this uses the same local file backend now, but lets you swap to azure.blob/s3 later
object.storage <- storage.init("object", provider = "file", base.dir = base.dir)
storage.set(object.storage, "name", "Roy Wetherall")
storage.get(object.storage, "name")

```

```

# set a default provider via options
old.options <- options(memofunc.storage.provider = list(name = "file", base.dir = base.dir))
on.exit(options(old.options), add = TRUE)
default.storage <- storage.init()
storage.get(default.storage, "name")

# Azure Blob provider example (requires AzureStor + credentials)
if (requireNamespace("AzureStor", quietly = TRUE)) {
  account <- Sys.getenv("AZURE_STORAGE_ACCOUNT")
  container <- Sys.getenv("AZURE_STORAGE_CONTAINER")
  key <- Sys.getenv("AZURE_STORAGE_KEY")
  token <- Sys.getenv("AZURE_STORAGE_TOKEN")

  if (account != "" && container != "" && (key != "" || token != "")) {
    azure.storage <- storage.init(
      "object",
      provider = "azure.blob",
      account = account,
      container = container,
      key = if (key == "") NULL else key,
      token = if (token == "") NULL else token,
      prefix = "memofunc-example"
    )

    storage.set(azure.storage, "name", "Roy Wetherall")
    storage.get(azure.storage, "name")
  }
}

```

---

storage.unset.object    *Unset a value that corresponds to a key within an object store.*

---

## Description

Unsets the value stored for a given key.

If there is no value for the key provided no action is taken.

## Usage

```
## S3 method for class 'object'
storage.unset(storage, key)
```

## Arguments

|         |                                |
|---------|--------------------------------|
| storage | initialized storage            |
| key     | key whose value is to be unset |

**Value**

Invisibly returns storage

**Examples**

```
library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")

# initialize file storage (direct backend; simplest local-only path)
base.dir <- file.path(tempdir(), "memofunc-storage")
file.storage <- storage.init("file", base.dir = base.dir)

# set and retrieve a value from file storage (direct backend)
storage.set(file.storage, "name", "Roy Wetherall")
storage.get(file.storage, "name")

# initialize object storage using a provider name (provider-backed interface)
```

```
# this uses the same local file backend now, but lets you swap to azure.blob/s3 later
object.storage <- storage.init("object", provider = "file", base.dir = base.dir)
storage.set(object.storage, "name", "Roy Wetherall")
storage.get(object.storage, "name")

# set a default provider via options
old.options <- options(memofunc.storage.provider = list(name = "file", base.dir = base.dir))
on.exit(options(old.options), add = TRUE)
default.storage <- storage.init()
storage.get(default.storage, "name")

# Azure Blob provider example (requires AzureStor + credentials)
if (requireNamespace("AzureStor", quietly = TRUE)) {
  account <- Sys.getenv("AZURE_STORAGE_ACCOUNT")
  container <- Sys.getenv("AZURE_STORAGE_CONTAINER")
  key <- Sys.getenv("AZURE_STORAGE_KEY")
  token <- Sys.getenv("AZURE_STORAGE_TOKEN")

  if (account != "" && container != "" && (key != "" || token != "")) {
    azure.storage <- storage.init(
      "object",
      provider = "azure.blob",
      account = account,
      container = container,
      key = if (key == "") NULL else key,
      token = if (token == "") NULL else token,
      prefix = "memofunc-example"
    )

    storage.set(azure.storage, "name", "Roy Wetherall")
    storage.get(azure.storage, "name")
  }
}
```

# Index

`functionCall`, [2](#)

`hash`, [3](#)

`hash.default`, [4](#)

`hash.function`, [5](#)

`hash.functionCall`, [6](#)

`hash.list`, [7](#)

`invalidate`, [8](#)

`is.memo`, [9](#)

`memo`, [9](#)

`memo.cache`, [13](#)

`memo.function`, [13](#)

`storage.clear`, [14](#)

`storage.clear.memory`, [16](#)

`storage.clear.object`, [18](#)

`storage.get`, [20](#)

`storage.get.memory`, [22](#)

`storage.get.object`, [24](#)

`storage.has`, [26](#)

`storage.has.memory`, [28](#)

`storage.has.object`, [30](#)

`storage.init`, [32](#)

`storage.init.memory`, [34](#)

`storage.init.object`, [36](#)

`storage.set`, [39](#)

`storage.set.memory`, [41](#)

`storage.set.object`, [43](#)

`storage.unset`, [45](#)

`storage.unset.memory`, [47](#)

`storage.unset.object`, [49](#)