

# Package ‘keylist’

April 27, 2026

**Title** Lightweight List Extensions that Enforce Unique Keys

**Version** 1.0.0

**Description** Provides two lightweight keylist S3 classes 'klist' and 'knlist': extensions of list that enforce unique keys, supporting either mixed named/unnamed elements or fully named elements, ensuring predictable key-value access.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**Suggests** testthat (>= 3.0.0)

**Config/testthat/edition** 3

**URL** <https://lj-jenkins.github.io/keylist/>,  
<https://github.com/LJ-Jenkins/keylist>

**BugReports** <https://github.com/LJ-Jenkins/keylist/issues>

**NeedsCompilation** yes

**Author** Luke Jenkins [aut, cre, cph] (ORCID:  
<<https://orcid.org/0000-0002-7206-7242>>)

**Maintainer** Luke Jenkins <luke-jenkins-dev@outlook.com>

**Repository** CRAN

**Date/Publication** 2026-04-27 14:20:02 UTC

## Contents

keylist . . . . .	2
klist . . . . .	3
knlist . . . . .	5
<b>Index</b>	<b>7</b>

---

keylist	<i>Create a List With Unique Keys</i>
---------	---------------------------------------

---

### Description

Create a list with unique keys (either fully named, `knlist`, or a mix of named and unnamed, `klist`), erroring if duplicate keys are found.

### Usage

```
keylist(..., .named = FALSE)

is.keylist(x)

as.keylist(x, ...)

## Default S3 method:
as.keylist(x, ..., .named = FALSE, .recursive = FALSE)

keylist.append(klst, values, after = length(klst))
```

### Arguments

<code>...</code>	for <code>keylist</code> : objects, possibly named, for <code>as.keylist</code> : passed to other methods.
<code>.named</code>	boolean indicating whether the list should be fully named or not. If <code>TRUE</code> then all elements must be named and unique, if <code>FALSE</code> then names are not required but if they are present they must be unique.
<code>x</code>	object to be coerced or tested.
<code>.recursive</code>	boolean indicating whether to recursively validate nested lists. If <code>TRUE</code> then all nested lists will be validated and converted to keylists, if <code>FALSE</code> then only the top level list will be validated and converted to a keylist.
<code>klst</code>	passed to <code>append</code> : the keylist to which the values are to be appended.
<code>values</code>	passed to <code>append</code> : the values to be included in the modified keylist.
<code>after</code>	passed to <code>append</code> : a subscript, after which the values are to be appended.

### Details

`keylist()` creates one of the two keylist objects: `klist` or `knlist`, depending on the value of the `.named` argument. Those methods are generally preferred as they are more explicit.

`is.keylist()` checks if an object inherits from either `klist` or `knlist`.

`keylist.append()` applies `append` to a keylist and validates the result depending on whether the input was a `klist` or `knlist`.

keylist objects accept any R object as elements, but the keys must be unique. The key validation is done at the top level, so nested lists are not validated unless they are also keylists. To recursively

turn a nested list structure into keylists, use `as.keylist(x, .recursive = TRUE)` or one of the lower level variants.

### Value

A list object of class `klist` or `knlist`. For `is.keylist()` a boolean.

### Note

keylists compare names using C's `strcmp` function.

[as.list](#) and [as.vector](#) methods for keylist objects remove the class and return a base R list or vector.

### See Also

[klist](#) and [knlist](#).

### Examples

```
keylist(a = 1, 2, b = 3) # default is a klist
try(keylist(1, a = 2, a = 1)) # duplicate keys not allowed

x <- keylist(a = 1, b = 2, .named = TRUE) # create a knlist
try(keylist(1, b = 2, .named = TRUE)) # unnamed keys not allowed
try(x[[1]] <- 1) # knlist only accepts character indexing for assignment

# objects within a keylist are not subject to validation
keylist(1, list(a = 1, a = 2))
try(keylist(1, keylist(a = 1, a = 2))) # but nested keylists are

# recursively validate and convert to keylist
x <- list(1, list(1, 2))
x <- as.keylist(x, .recursive = TRUE)
class(x[[2]]) # nested list is now a keylist

is.keylist(klist(1)) && is.keylist(knlist(a = 1)) # TRUE

keylist.append(klist(a = 1), list(2, b = 3)) # append to a klist
# c() method for keylist objects also validates
try(c(keylist(a = 1), list(a = 3)))
```

---

klist

*Create a List With Unique Keys (Named or Unnamed)*

---

### Description

Create a named/unnamed list with unique keys, erroring if any duplicate keys (names) are found.

**Usage**

```

klist(...)

is.klist(x)

as.klist(x, ...)

## Default S3 method:
as.klist(x, ..., .recursive = FALSE)

## S3 replacement method for class 'klist'
names(x) <- value

## S3 method for class 'klist'
c(...)
```

**Arguments**

...	for klist: objects, possibly named, for as.klist: passed to other methods.
x	object to be coerced or tested.
.recursive	boolean indicating whether to recursively validate nested lists. If TRUE then all nested lists will be validated and converted to klists, if FALSE then only the top level list will be validated and converted to a klist.
value	a character vector of up to the same length as x, or NULL. Resulting names must be unique.

**Details**

names<- method for klist objects will apply the names and then validate that the resulting names are unique.

c.klist method will combine the objects and then validate that the resulting klist's names are unique.

For a list with unique keys of all names, see [knlist](#).

**Value**

A list object of class klist. For is.klist() a boolean.

**Note**

klists compare names using C's strcmp function.

[as.list](#) and [as.vector](#) methods for klist objects remove the class and return a base R list or vector.

**See Also**

[knlist](#) and [keylist](#).

**Examples**

```

knlist(a = 1, 2, b = 3)
try(knlist(1, a = 2, a = 1)) # duplicate keys not allowed

# objects within a knlist are not subject to validation
knlist(1, list(a = 1, a = 2))
try(knlist(1, knlist(a = 1, a = 2))) # but nested knlists are

# recursively validate and convert to knlist
x <- list(1, list(1, 2))
x <- as.knlist(x, .recursive = TRUE)
class(x[[2]]) # nested list is now a knlist

is.knlist(knlist(1)) # TRUE

try(names(x) <- c("a", "a")) # names are validated when changed

# c() method for knlist objects also validates
try(c(knlist(a = 1), list(a = 3)))

```

---

knlist

---

*Create a List With Unique Named Keys*


---

**Description**

Create a list with unique names/keys, erroring if any duplicate names/keys are found.

**Usage**

```

knlist(...)

is.knlist(x)

as.knlist(x, ...)

## Default S3 method:
as.knlist(x, ..., .recursive = FALSE)

## S3 replacement method for class 'knlist'
names(x) <- value

## S3 method for class 'knlist'
c(...)

```

**Arguments**

```

...      for knlist: named objects, for as.knlist: passed to other methods.
x        object to be coerced or tested.

```

`.recursive`      boolean indicating whether to recursively validate nested lists. If TRUE then all nested lists will be validated and converted to knlists, if FALSE then only the top level list will be validated and converted to a knlist.

`value`            a character vector the same length as `x`. Resulting names must be unique.

### Details

`names<-` method for `knlist` objects will apply the names and then validate that the resulting names are unique.

`c.knlist` method will combine the objects and then validate that the resulting `knlist`'s names are unique.

For a list with unique keys of names and indexes, see [klist](#).

### Value

A list object of class `knlist`. For `is.knlist()` a boolean.

### Note

`knlists` compare names using C's `strcmp` function.

[as.list](#) and [as.vector](#) methods for `klist` objects remove the class and return a base R list or vector.

### See Also

[klist](#) and [keylist](#).

### Examples

```
x <- knlist(a = 1, b = 2, c = 3)
try(knlist(b = 1, a = 2, a = 1)) # duplicate keys not allowed
try(x[[1]] <- 1) # knlist only accepts character indexing for assignment

# objects within a knlist are not subject to validation
knlist(x = 1, y = list(a = 1, a = 2))
try(knlist(x = 1, y = knlist(a = 1, a = 2))) # but nested knlists are

# recursively validate and convert to knlist
x <- list(a = 1, b = list(x = 1, y = 2))
x <- as.knlist(x, .recursive = TRUE)
class(x[[2]]) # nested list is now a knlist

is.knlist(knlist(a = 1)) # TRUE

try(names(x) <- c("a", "a")) # names are validated when changed

# c() method for knlist objects also validates
try(c(knlist(a = 1), list(a = 3)))
```

# Index

append, 2  
as.keylist (keylist), 2  
as.klist (klist), 3  
as.knlist (knlist), 5  
as.list, 3, 4, 6  
as.vector, 3, 4, 6  
  
c.klist (klist), 3  
c.knlist (knlist), 5  
  
is.keylist (keylist), 2  
is.klist (klist), 3  
is.knlist (knlist), 5  
  
keylist, 2, 4, 6  
klist, 2, 3, 3, 6  
knlist, 2–4, 5  
  
names<- .klist (klist), 3  
names<- .knlist (knlist), 5