

# Package: dictionar6 (via r-universe)

February 17, 2025

**Title** R6 Dictionary Interface

**Description** Efficient object-oriented R6 dictionary capable of holding objects of any class, including R6. Typed and untyped dictionaries are provided as well as the 'usual' dictionary methods that are available in other OOP languages, for example listing keys, items, values, and methods to get/set these.

**Version** 0.1.3

**License** MIT + file LICENSE

**URL** <https://xoops.github.io/dictionar6/>,  
<https://github.com/xoops/dictionar6/>

**BugReports** <https://github.com/xoops/dictionar6/issues>

**Config/testthat/edition** 3

**Encoding** UTF-8

**NeedsCompilation** no

**Roxygen** list(markdown = TRUE, r6 = TRUE)

**RoxygenNote** 7.1.1

**Imports** ooplah, R6

**Suggests** testthat

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

**RemoteUrl** <https://github.com/xoops/dictionar6>

**RemoteRef** HEAD

**RemoteSha** 8a460ee0d9f6b628eda242ef8991176036667bc8

## Contents

dct . . . . .	2
Dictionary . . . . .	3
<b>Index</b>	<b>7</b>

---

dct	<i>Construct a Dictionary</i>
-----	-------------------------------

---

## Description

Sugar function wrapping `[Dictionary]$new`.

## Usage

```
dct(..., x = list(...), types = NULL)
```

## Arguments

<code>...</code>	(ANY) Named arguments with names corresponding to the items to add to the dictionary, where the keys are the names and the values are the elements. Names must be unique.
<code>x</code>	(list()) A named list with the names corresponding to the items to add to the dictionary, where the keys are the list names and the values are the list elements. Names must be unique.
<code>types</code>	(character()) If non-NULL then <code>types</code> creates a typed dictionary in which all elements of the dictionary must inherit from these types. Any class can be given to <code>types</code> as long as there is a valid <code>as.character</code> method associated with the class.

## Examples

```
# untyped
dct(a = 1, b = 2, c = "a")

# typed - class is forced
dct(a = 1L, b = 2L, c = 3L, types = "integer")

# list constructor
dct(x = list(a = 1, b = 2, c = "a"))

# with R6
d <- dct(a = 1)
dct(d = d)
```

**Description**

Dictionaries are essential objects in other object-oriented languages, such as Python. The primary function of a dictionary is to store items in a key-value pair. Where the key is the name of the item and the value is the value. This object contains all the 'usual' methods that would be found in other languages, including setting and getting values, adding and removing items, and containedness checks.

**Active bindings**

keys None -> character()  
Get dictionary keys.

values None -> list()  
Get dictionary values.

items list() -> self / None -> list()  
If x is missing then returns the dictionary items.  
If x is not missing then used to set items in the dictionary.

length None -> integer(1)  
Get dictionary length as number of items.

typed None -> logical(1)  
Get if the dictionary is typed (TRUE) or not (FALSE).

types None -> character()  
Get the dictionary types (NULL if un-typed).

**Methods****Public methods:**

- Dictionary\$new()
- Dictionary\$add()
- Dictionary\$rekey()
- Dictionary\$revalue()
- Dictionary\$remove()
- Dictionary\$get()
- Dictionary\$get\_list()
- Dictionary\$has()
- Dictionary\$assert\_contains()
- Dictionary\$has\_value()
- Dictionary\$print()
- Dictionary\$summary()
- Dictionary\$merge()

- [Dictionary\\$clone\(\)](#)

**Method new():** Constructs a Dictionary object.

*Usage:*

```
Dictionary$new(..., x = list(...), types = NULL)
```

*Arguments:*

... (ANY)

Named arguments with names corresponding to the items to add to the dictionary, where the keys are the names and the values are the elements. Names must be unique.

x (list())

A named list with the names corresponding to the items to add to the dictionary, where the keys are the list names and the values are the list elements. Names must be unique.

types (character())

If non-NULL then types creates a typed dictionary in which all elements of the dictionary must inherit from these types. Any class can be given to types as long as there is a valid as.character method associated with the class.

**Method add():** Add new items to the dictionary.

*Usage:*

```
Dictionary$add(x = list(), keys = NULL, values = NULL)
```

*Arguments:*

x (list())

Same as initialize, items to add to the list.

keys (character())

If x is NULL then keys and values can be provided to add the new items by a character vector of keys and list of values instead.

values (list())

If x is NULL then keys and values can be provided to add the new items by a list of keys and values instead.

**Method rekey():** Change the name of a given key.

*Usage:*

```
Dictionary$rekey(key, new_key)
```

*Arguments:*

key (character(1))

Key to rename.

new\_key (character(1))

New name of key, must not already exist in dictionary.

**Method revalue():** Change the value of a given item.

*Usage:*

```
Dictionary$revalue(key, new_value)
```

*Arguments:*

key (character(1))

Key of item to revalue.

new\_value (character(1))  
New value of item.

**Method** remove(): Removes the given item from the list.

*Usage:*

Dictionary\$remove(key)

*Arguments:*

key (character(1))  
Key of item to remove.

**Method** get(): Gets the given items from the dictionary. If only one item is requested then returns the (unlisted) item, or if multiple items are requested as the dictionary is typed, then the unlisted items are returned.

*Usage:*

Dictionary\$get(keys, clone = TRUE)

*Arguments:*

keys (character())  
Keys of items to get.  
clone (logical(1))  
If TRUE (default) then deep clones R6 objects if requested.

**Method** get\_list(): Gets the given items from the dictionary as list.

*Usage:*

Dictionary\$get\_list(keys, clone = TRUE)

*Arguments:*

keys (character())  
Keys of items to get.  
clone (logical(1))  
If TRUE (default) then deep clones R6 objects if requested.

**Method** has(): Checks if the given key is in the list, returns a logical.

*Usage:*

Dictionary\$has(key)

*Arguments:*

key (character(1))  
Key to check.

**Method** assert\_contains(): Asserts if the given keys are in the list, returns keys invisibly if assertion passes otherwise errors.

*Usage:*

Dictionary\$assert\_contains(keys)

*Arguments:*

keys (character())  
Keys to check.

**Method** `has_value()`: Checks if the given value is in the list, returns a logical.

*Usage:*

`Dictionary$has_value(value)`

*Arguments:*

`value` (ANY)

Value to check.

**Method** `print()`: Prints dictionary.

*Usage:*

`Dictionary$print(n = 2)`

*Arguments:*

`n` (integer(1))

Number of items to print on either side of ellipsis.

**Method** `summary()`: Summarises dictionary.

*Usage:*

`Dictionary$summary(n = 2)`

*Arguments:*

`n` (integer(1))

Number of items to print on either side of ellipsis.

**Method** `merge()`: Merges another dictionary, or list of dictionaries, into self.

*Usage:*

`Dictionary$merge(x)`

*Arguments:*

`x` (Dictionary(1) | list())

Dictionary or list of dictionaries to merge in, must have unique keys.

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

`Dictionary$clone(deep = FALSE)`

*Arguments:*

`deep` Whether to make a deep clone.

# Index

dct, [2](#)  
Dictionary, [3](#)