

# Package ‘mappings’

June 23, 2021

**Type** Package

**Version** 0.1

**Date** 2021-06-21

**Title** Functions for Transforming Categorical Variables

**URL** <https://github.com/benjaminrich/mappings>

**BugReports** <https://github.com/benjaminrich/mappings/issues>

**Description** Easily create functions to map between different sets of values,  
such as for re-labelling categorical variables.

**License** GPL-3

## Imports

**Suggests** testthat

**Language** en-US

**Encoding** UTF-8

**RoxygenNote** 7.1.1

**NeedsCompilation** no

**Author** Benjamin Rich [aut, cre, cph]

**Maintainer** Benjamin Rich <[mail@benjaminrich.net](mailto:mail@benjaminrich.net)>

**Repository** CRAN

**Date/Publication** 2021-06-23 06:30:05 UTC

## R topics documented:

<code>as.data.frame.mapping</code>	2
<code>cf</code>	2
<code>cut_mapping</code>	3
<code>domain</code>	4
<code>inverse</code>	4
<code>mapping</code>	5
<code>print.mapping</code>	6
<code>remap</code>	7
<code>text2mapping</code>	7

**Index****9**

`as.data.frame.mapping` *Convert a mapping to data.frame*

**Description**

The resulting `data.frame` has 2 columns: `mapsfrom`, and `mapsto`.

**Usage**

```
## S3 method for class 'mapping'
as.data.frame(x, ...)
```

**Arguments**

`x`                    A [mapping](#).  
`...`                 Ignored.

**Value**

A `data.frame`.

`cf`                    *Construct a factor from one or more vectors*

**Description**

A factor is constructed from one or more atomic vectors. If more than one atomic vector is supplied, then a compound value is constructed by concatenating the values together. The order of the levels is the natural order in which the values appear.

**Usage**

```
cf(x, ..., sep = ";")
```

**Arguments**

`x`                    An atomic vector.  
`...`                 Additional atomic vectors (optional).  
`sep`                 A character to use as a separator when forming a compound value (default `;`).

**Value**

A factor.

**Examples**

```
x <- c("A", "B", "A")
y <- c(2, 5, 7)
cf(x, y)
mapping(cf(x, y), c("X", "Y", "Z"))
```

---

cut_mapping	<i>Mapping from continuous to categorical</i>
-------------	---

---

**Description**

Mapping from continuous to categorical

**Usage**

```
cut_mapping(..., to = NULL, na = NA, ch.as.fact = TRUE)
```

**Arguments**

...	Passed to <code>cut()</code> .
to	Passed to <code>mapping()</code> .
na	Passed to <code>mapping()</code> .
ch.as.fact	Passed to <code>mapping()</code> .

**Value**

A function that cuts a numeric vector and maps the result.

**Examples**

```
x <- c(0, 10, 20, 30, Inf)
m <- cut_mapping(x, right=FALSE,
  to=c("0 to <10", "10 to <20", "20 to <30", ">= 30"))
print(m)
m(c(5, 27, 3, 10, 99))
```

---

domain	<i>Domain and codomain of a mapping.</i>
--------	--

---

**Description**

Domain and codomain of a mapping.

**Usage**

```
domain(x)
```

```
codomain(x)
```

**Arguments**

x            A [mapping](#).

**Value**

x A vector of the same type as we supplied when the [mapping](#) was created.

**Note**

These aren't the true domain and codomain in the mathematical sense; both can contain duplicates.

**Examples**

```
sex.mapping <- mapping(c("Female", "F", "Male", "M"), c(0, 0, 1, 1))
domain(sex.mapping)
codomain(sex.mapping)
```

---

inverse	<i>Inverse of a mapping</i>
---------	-----------------------------

---

**Description**

Given a [mapping](#) x, return the inverse mapping.

**Usage**

```
inverse(x)
```

**Arguments**

x            A [mapping](#).

**Value**

The inverse [mapping](#).

**Examples**

```
sex.mapping <- mapping(c("Female", "F", "Male", "M"), c(0, 0, 1, 1))
sex.inverse.mapping <- inverse(sex.mapping)
sex.inverse.mapping(c(0, 0, 1, 0))
```

---

mapping	<i>Generate a Mapping Function</i>
---------	------------------------------------

---

**Description**

This function returns a function that does a simple mapping from one set of value to another. It is a function-generating function.

**Usage**

```
mapping(from, to, na = NA, ch.as.fact = TRUE)
```

**Arguments**

from	A vector. This is the domain of the function.
to	A vector of the same length as from. If omitted, then the names of from are taken as the domain, and the values as the values to map to. If from has no names, then to is equal to from (useful for re-ordering factor levels).
na	An alternative way to specify the value that NA maps to. Ignored if from contains NA.
ch.as.fact	A logical. Should the mapping return a factor instead of character?

**Details**

This function returns a function. When called with a vector argument  $x$ , this function will return a vector  $y$  of the same length as  $x$  and such that each element  $y[i]$  is equal to  $to[j]$  where  $j$  is the smallest integer such that  $from[j] == x[i]$ , and NA if no such  $j$  exists.

Note: from will always be matched as a string, even if it is numeric. So, `mapping(1, "A")` and `mapping("1", "A")` are the same, and both functions will return "A" when called with either 1 or "1".

**Value**

A function that translates from from to to. The function also has an [inverse](#) which is a function that performs the inverse mapping.

**See Also**

[inverse\(\)](#), [codomain\(\)](#), [domain\(\)](#), [remap\(\)](#), [text2mapping\(\)](#), [cut\\_mapping\(\)](#)

**Examples**

```
sex.mapping <- mapping(c("Female", "F", "Male", "M"), c(0, 0, 1, 1))
sex.mapping(c("Female", "Female", "Male", "F"))
```

```
sex.mapping <- mapping(0:1, c("Female", "Male"), na="Unknown")
sex.mapping(c(0, 1, NA, 0, 1, 1, 0))
inverse(sex.mapping)(c("Female", "Male", "Unknown"))
```

```
from <- c(0, 1, NA)
to <- c(NA, "Male", "Female")
x <- c(0, 1, NA, 0, 1, 1, 0)
sex.mapping <- mapping(c(0, 1, NA), c(NA, "Male", "Female"))
sex.mapping
sex.mapping(c(0, 1, NA, 0, 1, 1, 0))
inverse(sex.mapping)
inverse(sex.mapping)(c("Female", "Male", NA))
```

```
race.mapping <- mapping(c(
  "1"="WHITE",
  "2"="BLACK OR AFRICAN AMERICAN",
  "5"="AMERICAN INDIAN OR ALASKA NATIVE"))
race.mapping(1:5)
```

---

print.mapping

*Print a mapping*

---

**Description**

Print a mapping

**Usage**

```
## S3 method for class 'mapping'
print(x, ...)
```

**Arguments**

x                    [mapping](#).  
 ...                  Ignored.

**Value**

Returns x invisibly.

---

remap	<i>Re-map a variable</i>
-------	--------------------------

---

**Description**

Apply a mapping to a vector directly. The mapping is temporary and not saved.

**Usage**

```
remap(x, ...)
```

**Arguments**

x	The values to apply the <a href="#">mapping</a> to.
...	Passed to <a href="#">mapping()</a> .

**Value**

The values returned by calling the [mapping](#) function.

**Examples**

```
x <- c("A", "B", "A")
remap(x, c(A=0, B=1))
```

---

text2mapping	<i>Convenient shorthand for specifying mappings with text strings</i>
--------------	---

---

**Description**

Convenient shorthand for specifying mappings with text strings

**Usage**

```
text2mapping(  
  text,  
  file = NULL,  
  sep = "|",  
  flip = FALSE,  
  convert.na = TRUE,  
  numericWherePossible = TRUE,  
  ...  
)
```

**Arguments**

text	A multi-line string specifying a mapping with 2 columns (see examples).
file	If text is missing, read from this file instead.
sep	Character used as column separator.
flip	If TRUE, flip the column order to To, From (default FALSE).
convert.na	If TRUE, the string "NA" will be converted to NA.
numericWherePossible	If TRUE, the mapping will return a numeric vector if the codomain contains only numbers.
...	Further arguments passed to <code>mapping()</code> .

**Value**

A `mapping`.

**Examples**

```
f <- text2mapping("
L | Low
M | Medium
H | High
")
f(warpbreaks$tension)
```

# Index

`as.data.frame.mapping`, 2

`cf`, 2

`codomain(domain)`, 4

`codomain()`, 6

`cut()`, 3

`cut_mapping`, 3

`cut_mapping()`, 6

`domain`, 4

`domain()`, 6

`inverse`, 4, 5

`inverse()`, 6

`mapping`, 2, 4, 5, 5, 6–8

`mapping()`, 3, 7, 8

`print.mapping`, 6

`remap`, 7

`remap()`, 6

`text2mapping`, 7

`text2mapping()`, 6