

# Package ‘qtl2convert’

July 1, 2020

**Version** 0.22-7

**Date** 2020-06-30

**Title** Convert Data among QTL Mapping Packages

**Description** Functions to convert data structures among the 'qtl2', 'qtl', and 'DOQTL' packages for mapping quantitative trait loci (QTL).

**Author** Karl W Broman [aut, cre] (<<https://orcid.org/0000-0002-4914-6671>>)

**Maintainer** Karl W Broman <[broman@wisc.edu](mailto:broman@wisc.edu)>

**Depends** R (>= 3.1.0)

**Imports** Rcpp (>= 0.12.12), qtl (>= 1.40-8), qtl2 (>= 0.18), utils, stats

**Suggests** testthat, devtools, roxygen2

**License** GPL-3

**URL** <http://kbroman.org/qtl2>, <https://github.com/rqtl/qtl2convert>

**BugReports** <https://github.com/rqtl/qtl2convert/issues>

**LinkingTo** Rcpp

**LazyData** true

**Encoding** UTF-8

**ByteCompile** true

**RoxygenNote** 7.1.1

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2020-07-01 08:40:07 UTC

## R topics documented:

<code>cbind_smother</code> . . . . .	2
<code>count_unique_geno</code> . . . . .	3
<code>encode_geno</code> . . . . .	3
<code>find_consensus_geno</code> . . . . .	4

find_unique_geno . . . . .	5
map_df_to_list . . . . .	6
map_list_to_df . . . . .	7
probs_doqtl_to_qtl2 . . . . .	8
probs_qtl2_to_array . . . . .	9
probs_qtl2_to_doqtl . . . . .	9
probs_qtl_to_qtl2 . . . . .	10
scan_qtl2_to_qtl . . . . .	11
scan_qtl_to_qtl2 . . . . .	12
write2csv . . . . .	12

<b>Index</b>	<b>15</b>
--------------	-----------

---

cbind_smother	<i>Combine matrices by columns, replacing matching ones and adding unique ones</i>
---------------	--

---

## Description

This is like `base::cbind()` but if a column in the second matrix has the same name as a column in the first matrix, the column in the first matrix is deleted and that in the second matrix is used in its place.

## Usage

```
cbind_smother(mat1, mat2)
```

## Arguments

mat1	A matrix
mat2	Another matrix, with the same number of rows as mat.

## Value

The two matrices combined by columns, but columns in the first matrix that also appear in the second matrix are deleted and replaced by those in the second matrix. Uses the row names to align the rows in the two matrices, and to expand them as needed.

## Examples

```
df1 <- data.frame(x=c(1,2,3,NA,4), y=c(5,8,9,10,11), row.names=c("A", "B", "C", "D", "E"))
df2 <- data.frame(z=c(7,8,0,9,10), y=c(6,NA,NA,9,10), row.names=c("A", "B", "F", "C", "D"))
df1n2 <- cbind_smother(df1, df2)
```

---

count_unique_genotype	<i>Count the unique genotypes for each row of a genotype matrix</i>
-----------------------	---

---

### Description

For genotype data (markers x individuals) on a set of individuals, count the unique genotypes for each marker

### Usage

```
count_unique_genotype(genotypes, na.strings = c("N", "H", "NA", ""), cores = 1)
```

### Arguments

genotypes	Matrix of genotypes (markers x individuals)
na.strings	Genotypes to be considered as missing values.
cores	Number of CPU cores to use, for parallel calculations. (If 0, use <a href="#">parallel::detectCores()</a> .) Alternatively, this can be links to a set of cluster sockets, as produced by <a href="#">parallel::makeCluster()</a> .

### Value

Vector of counts of unique genotypes.

### See Also

[find\\_unique\\_genotype\(\)](#)

### Examples

```
g <- rbind(c("NA", "A", "A", "A", "T"),
           c("NA", "NA", "NA", "A", "A"),
           c("A", "A", "T", "G", "G"),
           c("C", "C", "G", "G", "NA"))
counts <- count_unique_genotype(g)
```

---

encode_genotype	<i>Encode a matrix of genotypes using a set of allele codes</i>
-----------------	---

---

### Description

Encode a matrix of genotypes using a set of allele codes.

**Usage**

```

encode_genotype(
  genotype,
  allele_codes,
  output_codes = c("-", "A", "H", "B"),
  cores = 1
)

```

**Arguments**

genotype	Character matrix of genotypes (rows as markers, columns as individuals)
allele_codes	Two-column matrix of alleles (rows as markers)
output_codes	Vector of length four, with missing, AA, AB, BB codes
cores	Number of CPU cores to use, for parallel calculations. (If 0, use <code>parallel::detectCores()</code> .) Alternatively, this can be links to a set of cluster sockets, as produced by <code>parallel::makeCluster()</code> .

**Value**

Matrix of same dimensions as genotype, but with values in output\_codes.

**See Also**

[find\\_consensus\\_genotype\(\)](#), [find\\_unique\\_genotype\(\)](#)

**Examples**

```

genotype <- rbind(c("C", "G", "C", "GG", "CG"),
                 c("A", "A", "AT", "TA", "TT"),
                 c("T", "G", NA, "GT", "TT"))
codes <- rbind(c("C", "G"), c("A", "T"), c("T", "G"))
genotype_encoded <- encode_genotype(genotype, codes)

```

---

find\_consensus\_genotype *Find the consensus genotype for each row of a genotype matrix*

---

**Description**

For genotype data (markers x individuals) on a set of individuals from a single inbred line, find the consensus genotype at each marker.

**Usage**

```

find_consensus_genotype(genotypes, na.strings = c("N", "H", "NA", ""), cores = 1)

```

**Arguments**

genotypes	Matrix of genotypes (markers x individuals)
na.strings	Genotypes to be considered as missing values.
cores	Number of CPU cores to use, for parallel calculations. (If 0, use <code>parallel::detectCores()</code> .) Alternatively, this can be links to a set of cluster sockets, as produced by <code>parallel::makeCluster()</code> .

**Value**

Vector of consensus genotypes, one value per row of genotypes

**See Also**

`find_unique_genotype()`, `encode_genotype()`

**Examples**

```
g <- rbind(c("NA", "N", "A", "A", "T", "G", NA, "H"),
          c("C", "C", "G", "G", "A", NA, NA, NA),
          rep(NA, 8),
          c("C", "C", "G", "G", "G", "C", "G", "G"))
consensus <- find_consensus_genotype(g)
```

---

find_unique_genotype	<i>Find the unique genotypes for each row of a genotype matrix</i>
----------------------	--

---

**Description**

For genotype data (markers x individuals) on a set of individuals, find the unique genotypes for each marker, provided that there are exactly two. (If more than two or fewer than two, return NAs.)

**Usage**

```
find_unique_genotype(genotypes, na.strings = c("N", "H", "NA", ""), cores = 1)
```

**Arguments**

genotypes	Matrix of genotypes (markers x individuals)
na.strings	Genotypes to be considered as missing values.
cores	Number of CPU cores to use, for parallel calculations. (If 0, use <code>parallel::detectCores()</code> .) Alternatively, this can be links to a set of cluster sockets, as produced by <code>parallel::makeCluster()</code> .

**Value**

Matrix with two columns. Each row corresponds to a marker, and has the two unique genotypes, or NAs (if >2 or <2 unique genotypes).

**See Also**

[count\\_unique\\_geno\(\)](#), [encode\\_geno\(\)](#)

**Examples**

```
g <- rbind(c("NA", "A", "A", "A", "T"),
           c("NA", "NA", "NA", "A", "A"),
           c("A", "A", "T", "G", "G"),
           c("C", "C", "G", "G", "NA"))
ug <- find_unique_geno(g)
```

---

map\_df\_to\_list

*Marker map data frame to list*

---

**Description**

Convert a marker map organized as data frame to a list

**Usage**

```
map_df_to_list(
  map,
  chr_column = "chr",
  pos_column = "cM",
  marker_column = "marker",
  Xchr = c("X", "X")
)
```

**Arguments**

map	Data frame with marker map
chr_column	Name of the column in map that contains the chromosome IDs.
pos_column	Name of the column in map that contains the marker positions.
marker_column	Name of the column in map that contains the marker names. If NULL, use the row names.
Xchr	Vector of character strings indicating the name or names of the X chromosome. If NULL, assume there's no X chromosome.

**Value**

A list of vectors of marker positions, one component per chromosome

**See Also**

[map\\_list\\_to\\_df\(\)](#)

**Examples**

```
map <- data.frame(chr=c(1,1,1, 2,2,2, "X","X"),
                 pos=c(0,5,10, 0,8,16, 5,20),
                 marker=c("D1M1","D1M2","D1M3", "D2M1","D2M2","D2M3", "DXM1","DXM2"))
map_list <- map_df_to_list(map, pos_column="pos")
```

---

map\_list\_to\_df                      *Marker map list to data frame*

---

**Description**

Convert a marker map organized as a list to a data frame

**Usage**

```
map_list_to_df(
  map_list,
  chr_column = "chr",
  pos_column = "pos",
  marker_column = "marker"
)
```

**Arguments**

map\_list                      List of vectors containing marker positions  
chr\_column                    Name of the chromosome column in the output  
pos\_column                    Name of the position column in the output  
marker\_column                Name of the marker column in the output. If NULL, just put them as row names.

**Value**

A data frame with the marker positions.

**See Also**

[map\\_df\\_to\\_list\(\)](#)

**Examples**

```
library(qtl2)
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
iron_map <- map_list_to_df(iron$gmap)
```

---

probs\_doqtl1\_to\_qtl2    *Convert DOQTL genotype probabilities to R/qtl2 format*

---

### Description

Convert DOQTL genotype probabilities to R/qtl2 format

### Usage

```
probs_doqtl1_to_qtl2(  
  probs,  
  map,  
  is_female = NULL,  
  chr_column = "chr",  
  pos_column = "cM",  
  marker_column = "marker"  
)
```

### Arguments

probs	3d array of genotype probabilities as calculated from DOQTL (individuals x genotypes x positions)
map	Data frame with marker map
is_female	Optional logical vector indicating which individuals are female. Names should contain the individual identifiers, matching the row names in probs.
chr_column	Name of the column in map that contains the chromosome IDs.
pos_column	Name of the column in map that contains the marker positions.
marker_column	Name of the column in map that contains the marker names. If NULL, use the row names.

### Details

We assume that the X chromosome is labeled "X" (must be upper-case) and that any other chromosomes are autosomes. We assume that the genotypes are labeled using the 8 letters A-H.

If the probabilities are for the full 36 states and the X chromosome is included but `is_female` is not provided, we'll guess which individuals are females based on their genotype probabilities. (If the average, across loci, of the sum of the heterozygote probabilities is small, we'll assume it's a female.)

### Value

An object of the form produced by `qtl2::calc_genoprob()`.



---

probs\_qt12\_to\_array    *Convert R/qt12 genotype probabilities to a 3d array*

---

**Description**

Convert R/qt12 genotype probabilities to a 3d array

**Usage**

```
probs_qt12_to_array(probs)
```

**Arguments**

probs            A "calc\_genoprob" object (a list of 3d arrays of genotype probabilities), as calculated by `qt12::calc_genoprob()`.

**Details**

We convert just the autosomal genotype probabilities, because they should all have the same number of genotypes (columns). The main application of this is for identifying possible sample mix-ups among batches of genotype probabilities (e.g., using the [R/lineup2](#) package), and for this the autosomal genotype probabilities should be sufficient.

**Value**

A single three-dimensional array, with just the autosomal genotype probabilities.

---

probs\_qt12\_to\_doqtl    *Convert R/qt12 genotype probabilities to DOQTL format*

---

**Description**

Convert R/qt12 genotype probabilities to DOQTL format

**Usage**

```
probs_qt12_to_doqtl(probs)
```

**Arguments**

probs            A "calc\_genoprob" object (a list of 3d arrays of genotype probabilities), as calculated by `qt12::calc_genoprob()`.

**Details**

If the arrays in probs all have 8 columns, they're assumed to be allele dosages and we paste them all together into one big array.

Otherwise, it should be that the autosomes all have 36 columns the X chromosome has 44. In this case, the male hemizygotes on the X are placed where the female homozygotes are, and then we reorder the genotypes into alphabetical order.

**Value**

A single three-dimensional array, for use with **DOQTL**.

---

probs_qtl_to_qtl2	<i>Convert R/qtl genotype probabilities to R/qtl2 format</i>
-------------------	--

---

**Description**

Convert R/qtl genotype probabilities to R/qtl2 format

**Usage**

```
probs_qtl_to_qtl2(cross)
```

**Arguments**

**cross** An R/qtl "cross" object (see [qtl::read.cross\(\)](#) for details.) Must contain genotype probabilities as calculated by [qtl::calc.genoprob\(\)](#).

**Value**

A list with two components:

- "probs" - the genotype probabilities in the form produced by [qtl2::calc\\_genoprob\(\)](#)
- "map" - Map of marker/pseudomarker positions (a list of vectors of positions)

**Examples**

```
library(qtl)
data(hyper)
hyper <- calc.genoprob(hyper, step=1, error.prob=0.002)
result <- probs_qtl_to_qtl2(hyper)
pr <- result$probs
map <- result$map
```

---

scan_qtl2_to_qtl	<i>Convert scan1 results to the scanone format</i>
------------------	--

---

### Description

Convert the results of `qtl2::scan1()` to the form used by the R/qtl function `qtl::scanone()`.

### Usage

```
scan_qtl2_to_qtl(scan1_output, map)
```

### Arguments

`scan1_output` Matrix of LOD scores, as calculated by `qtl2::scan1()`.  
`map` Map of markers/pseudomarkers (as a list of vectors).

### Value

A data frame with class "scanone", containing chromosome and position columns followed by the LOD scores in `scan1_output`.

### See Also

[scan\\_qtl2\\_to\\_qtl\(\)](#)

### Examples

```
library(qtl2)
iron <- read_cross2(system.file("extdata", "iron.zip", package="qtl2"))
map <- insert_pseudomarkers(iron$gmap, step=1)
probs <- calc_genoprob(iron, map, error_prob=0.002)
pheno <- iron$pheno
covar <- match(iron$covar$sex, c("f", "m")) # make numeric
names(covar) <- rownames(iron$covar)
Xcovar <- get_x_covar(iron)
out <- scan1(probs, pheno, addcovar=covar, Xcovar=Xcovar)

out_rev <- scan_qtl2_to_qtl(out, map)
```

---

scan_qt1_to_qt12	<i>Convert R/ql scanone results to R/ql2 scan1 format</i>
------------------	---

---

**Description**

Convert the results of R/ql1 `qt1::scanone()` to the form used by the R/ql2 `qt12::scan1()`.

**Usage**

```
scan_qt1_to_qt12(scanone_output)
```

**Arguments**

`scanone_output` Data frame as output by the R/ql1 function `qt1::scanone()`.

**Value**

List with two objects: the LOD scores in `qt12::scan1()` format, and the map (as a list of marker/pseudomarker positions).

**See Also**

[scan\\_qt1\\_to\\_qt12\(\)](#)

**Examples**

```
library(qt1)
data(hyper)
hyper <- calc.genoprob(hyper, step=1, error.prob=0.002)
out <- scanone(hyper)
out2 <- scan_qt1_to_qt12(out)
```

---

write2csv	<i>Write a data frame to a CSV file</i>
-----------	---

---

**Description**

Write a data frame to a CSV file in a special form, with info about the number of rows and columns.

**Usage**

```
write2csv(
  df,
  filename,
  comment = "",
  sep = ",",
  comment.char = "#",
  row.names = NULL,
  overwrite = FALSE
)
```

**Arguments**

<code>df</code>	A data frame (or matrix)
<code>filename</code>	File name to write
<code>comment</code>	Comment to place on the first line
<code>sep</code>	Field separator
<code>comment.char</code>	Character to use to initiate the comment lines
<code>row.names</code>	If NA or NULL (the default), row names are not included in the output file. Otherwise, the row names are included as the first column of the output, and this is taken to be the name for that column.
<code>overwrite</code>	If TRUE, overwrite file if it exists. If FALSE (the default) and the file exists, stop with an error.

**Details**

If the file already exists, the function will refuse to write over it.

The file will include comments at the top, using # as a comment character, including the number of rows (not including the header) and the number of columns.

By default, row names are not included. But with the option `row.names` provided as a character string, they are added as an initial column, with the value of this argument defining the name for that column. If a column with that name already exists, the function halts with an error.

**Value**

None.

**Examples**

```
nr <- 10
nc <- 5
x <- data.frame(id=paste0("ind", 1:nr),
                matrix(rnorm(nr*nc), ncol=nc))
colnames(x)[1:nc + 1] <- paste0("col", 1:nc)

testfile <- file.path(tempdir(), "tmpfile.csv")
write2csv(x, testfile, "A file created by write2csv")
```

```
# Remove the file, to clean up temporary directory  
unlink(testfile)
```

# Index

`base::cbind()`, [2](#)

`cbind_smother`, [2](#)  
`count_unique geno`, [3](#)  
`count_unique geno()`, [6](#)

`encode geno`, [3](#)  
`encode geno()`, [5, 6](#)

`find_consensus geno`, [4](#)  
`find_consensus geno()`, [4](#)  
`find_unique geno`, [5](#)  
`find_unique geno()`, [3-5](#)

`map_df_to_list`, [6](#)  
`map_df_to_list()`, [7](#)  
`map_list_to_df`, [7](#)  
`map_list_to_df()`, [6](#)

`parallel::detectCores()`, [3-5](#)  
`parallel::makeCluster()`, [3-5](#)  
`probs_doqtl_to_qtl2`, [8](#)  
`probs_qtl2_to_array`, [9](#)  
`probs_qtl2_to_doqtl`, [9](#)  
`probs_qtl_to_qtl2`, [10](#)

`qtl2::calc_genoprob()`, [8-10](#)  
`qtl2::scan1()`, [11, 12](#)  
`qtl::calc.genoprob()`, [10](#)  
`qtl::read.cross()`, [10](#)  
`qtl::scanone()`, [11, 12](#)

`scan_qtl2_to_qtl`, [11](#)  
`scan_qtl2_to_qtl()`, [11](#)  
`scan_qtl_to_qtl2`, [12](#)  
`scan_qtl_to_qtl2()`, [12](#)

`write2csv`, [12](#)