

# Package ‘csvread’

January 5, 2015

**Title** Fast Specialized CSV File Loader.

**Version** 1.2

**Author** Sergei Izrailev

**Maintainer** Sergei Izrailev <sizrailev@jabiruventures.com>

**Description** This package provides functions for loading large (10M+ lines) CSV and other delimited files, similar to read.csv, but typically faster and using less memory than the standard R loader. While not entirely general, it covers many common use cases when the types of columns in the CSV file are known in advance. In addition, the package provides a class 'int64', which represents 64-bit integers exactly when reading from a file. The latter is useful when working with 64-bit integer identifiers exported from databases. The CSV file loader supports common column types including 'integer', 'double', 'string', and 'int64', leaving further type transformations to the user.

**URL** <http://github.com/collectivemedia/csvread>

**Depends** R (>= 2.15),  
methods

**Enhances** bit64

**License** Apache License (== 2.0)

**Copyright** Copyright (C) Collective, Inc. | file inst/COPYRIGHTS

**LazyData** true

## R topics documented:

csvread . . . . .	2
int64 . . . . .	6
Ops.int64 . . . . .	7
<b>Index</b>	<b>9</b>

---

 csvread

*Fast Specialized CSV File Loader.*


---

## Description

Package `csvread` contains a fast specialized CSV and other delimited file loader, and a basic 64-bit integer class to aid in reading 64-bit integer values.

Given a list of the column types, function `csvread` parses the CSV file and returns a data frame.

`map.coltypes` guesses the column types in the CSV file by reading the first `nrows` lines. The result can be passed to `csvread` as the `coltypes` argument.

## Usage

```
csvread(file, coltypes, header, colnames = NULL, nrows = NULL,
        verbose = FALSE, delimiter = ",", na.strings = c("NA", "na", "NULL",
        "null", ""))
```

```
map.coltypes(file, header, nrows = 100, delimiter = ",")
```

## Arguments

<code>file</code>	Path to the CSV file.
<code>coltypes</code>	<p>A vector of column types, e.g., <code>c("integer", "string")</code>. The accepted types are "integer", "double", "string", "long" and "longhex".</p> <ul style="list-style-type: none"> <li><code>integer</code> - the column is parsed into an R integer type (32 bit)</li> <li><code>double</code> - the column is parsed into an R double type</li> <li><code>string</code> - the column is loaded as character type</li> <li><code>long</code> - the column is interpreted as the decimal representation of a 64-bit integer, stored as a double and assigned the <code>int64</code> class.</li> <li><code>longhex</code> - the column is interpreted as the hex representation of a 64-bit integer, stored as a double and assigned the <code>int64</code> class with an additional attribute <code>base = 16L</code> that is used for printing.</li> <li><code>integer64</code> - same as <code>long</code> but produces a column of class <code>integer64</code>, which should be compatible with package <code>bit64</code> (untested).</li> <li><code>verbose</code> - if <code>TRUE</code>, the function prints number of lines counted in the file.</li> <li><code>delimiter</code> - a single character delimiter, default is <code>","</code>.</li> </ul>
<code>header</code>	<code>TRUE</code> (default) or <code>FALSE</code> ; indicates whether the file has a header and serves as the source of column names if <code>colnames</code> is not provided.
<code>colnames</code>	Optional column names for the resulting data frame. Overrides the header, if header is present. If <code>NULL</code> , then the column names are taken from the header, or, if there is no header, the column names are set to <code>'COL1'</code> , <code>'COL2'</code> , etc.
<code>nrows</code>	If <code>NULL</code> , the function first counts the lines in the file. This step can be avoided if the number of lines is known by providing a value to <code>nrows</code> . On the other hand, <code>nrows</code> can be used to read only the first lines of the CSV file.
<code>verbose</code>	If <code>TRUE</code> and <code>nrows</code> is <code>NULL</code> , the function prints number of lines counted in the file.
<code>delimiter</code>	A single character delimiter, default is <code>","</code> .
<code>na.strings</code>	A vector of strings to be considered NA in the input file.

## Details

csvread provides functionality for loading large (10M+ lines) CSV and other delimited files, similar to read.csv, but typically faster and using less memory than the standard R loader. While not entirely general, it covers many common use cases when the types of columns in the CSV file are known in advance. In addition, the package provides a class 'int64', which represents 64-bit integers exactly when reading from a file. The latter is useful when working with 64-bit integer identifiers exported from databases. The CSV file loader supports common column types including integer, double, string, and int64, leaving further type transformations to the user.

If number of columns, which is inferred from the number of provided coltypes, is greater than the actual number of columns, the extra columns are still created. If the number of columns is less than the actual number of columns in the file, the extra columns in the file are ignored. Commas included in double quotes will be considered part of the field, rather than a separator, but double quotes will NOT be stripped. Runaway double quotes will end at the end of the line.

See also [int64](#) for information about dealing with 64-bit integers when loading data from CSV files.

## Value

A data frame containing the data from the CSV file.

## Maintainer

Sergei Izrailev

## Copyright

Copyright (C) Collective, Inc.; with portions Copyright (C) Jabiru Ventures LLC

## License

Apache License, Version 2.0, available at <http://www.apache.org/licenses/LICENSE-2.0>

## URL

<http://github.com/jabiru/csvread>

## Installation from github

```
devtools::install_github("jabiru/csvread")
```

## Author(s)

Sergei Izrailev

## See Also

[int64](#)

**Examples**

```
## Not run:
frm <- csvread("inst/10rows.csv",
  coltypes = c("longhex", "string", "double", "integer", "long"),
  header = FALSE, nrows = 10)
frm
#           COL1          COL2          COL3 COL4 COL5
# 1  11fb89c1558c792 2011-05-06 0.150001 4970 4977
# 2  11fb89c1558c792 2011-05-06 0.150001 4970 4987
# 3  11fb89c1558c792 2011-05-06 0.150001 5200 5528
# 4  11fb89c1558c792 2011-05-06 0.150001 4970 5004
# 5  11fb89c1558c792 2011-05-06 0.150001 4970 4980
# 6  11fb89c1558c792 2011-05-06 0.150001 4970 5020
# 7  11fb89c1558c792 2011-05-06 0.150001 4970 5048
# 8  11fb89c1558c792 2011-05-06 0.150001 4970 5035
# 9  11fb89c1558c792 2011-05-06 0.150001 4970 4971
# 10 11fb89c1558c792 2011-05-06 0.150001 4970 4973

typeof(frm$COL1)
# [1] "double"
class(frm$COL1)
# [1] "int64"

typeof(frm$COL5)
# [1] "double"
class(frm$COL5)
# [1] "int64"

frm <- csvread("inst/10rows_na.csv",
  coltypes = c("longhex", "string", "double", "integer", "long"),
  header = FALSE, nrows = 10)
frm
#           COL1          COL2          COL3 COL4 COL5
# 1  11fb89c1558c792 2011-05-06 0.150001 4970 4977
# 2  11fb89c1558c792 2011-05-06 0.150001 4970 4987
# 3  11fb89c1558c792          NA 0.150001    NA 5528
# 4          <NA> 2011-05-06 0.150001 4970 5004
# 5  11fb89c1558c792          <NA> 0.150001 4970 4980
# 6  11fb89c1558c792 2011-05-06          NA 4970 5020
# 7  11fb89c1558c792 2011-05-06 0.150001    NA 5048
# 8  11fb89c1558c792 2011-05-06 0.150001 4970    NA
# 9          <NA> 2011-05-06 0.150001 4970 4971
# 10 11fb89c1558c792 2011-05-06 0.150001 4970    NA

## End(Not run)
## Not run:
coltypes <- map.coltypes("inst/10rows.csv", header = FALSE)
coltypes
#           V1          V2          V3          V4          V5
# "string"  "string"  "double" "integer" "integer"

frm <- csvread(file = "inst/10rows.csv", coltypes = coltypes, header = F, verbose = T)
# Counted 10 lines.

frm
```

```

#           COL1          COL2          COL3 COL4 COL5
# 1  11fb89c1558c792  2011-05-06  0.150001  4970  4977
# 2  11fb89c1558c792  2011-05-06  0.150001  4970  4987
# 3  11fb89c1558c792  2011-05-06  0.150001  5200  5528
# 4  11fb89c1558c792  2011-05-06  0.150001  4970  5004
# 5  11fb89c1558c792  2011-05-06  0.150001  4970  4980
# 6  11fb89c1558c792  2011-05-06  0.150001  4970  5020
# 7  11fb89c1558c792  2011-05-06  0.150001  4970  5048
# 8  11fb89c1558c792  2011-05-06  0.150001  4970  5035
# 9  11fb89c1558c792  2011-05-06  0.150001  4970  4971
# 10 11fb89c1558c792  2011-05-06  0.150001  4970  4973
typeof(frm$COL1)
# [1] "character"
class(frm$COL1)
# [1] "character"

typeof(frm$COL5)
# [1] "integer"
class(frm$COL5)
# [1] "integer"

frm$COL1 <- as.int64(frm$COL1, base = 16)
frm$COL1
# [1] "11fb89c1558c792" "11fb89c1558c792" "11fb89c1558c792" "11fb89c1558c792"
# [5] "11fb89c1558c792" "11fb89c1558c792" "11fb89c1558c792" "11fb89c1558c792"
# [9] "11fb89c1558c792" "11fb89c1558c792"
typeof(frm$COL1)
# [1] "double"
class(frm$COL1)
# [1] "int64"

as.character.int64(frm$COL1[1], base = 10)
# [1] "80986298828507026"

#####
# A file with NAs and missing values: note that the in the first column,
# an empty string in row 9 is not considered NA because na.strings are set to "NA".
# By default, the empty string will be considered NA. Also, in column 2,
# row 3, the value is " NA " (with spaces), which doesn't match the na.strings value
# and therefore is not considered an NA.
frm <- csvread(file = "inst/10rows.csv", coltypes = coltypes, header = F,
verbose = T, na.strings = "NA")
# Counted 10 lines.

frm
#           COL1          COL2          COL3 COL4 COL5
# 1  11fb89c1558c792  2011-05-06  0.150001  4970  4977
# 2  11fb89c1558c792  2011-05-06  0.150001  4970  4987
# 3  11fb89c1558c792           NA  0.150001   NA  5528
# 4           <NA>  2011-05-06  0.150001  4970  5004
# 5  11fb89c1558c792           <NA>  0.150001  4970  4980
# 6  11fb89c1558c792  2011-05-06           NA  4970  5020
# 7  11fb89c1558c792  2011-05-06  0.150001   NA  5048
# 8  11fb89c1558c792  2011-05-06  0.150001  4970   NA
# 9           2011-05-06  0.150001  4970  4971
# 10 11fb89c1558c792  2011-05-06  0.150001  4970   NA

```

```
## End(Not run)
```

---

int64	<i>A very basic 64-bit integer class.</i>
-------	---

---

## Description

A very basic 64-bit integer class.

## Usage

```
int64(length = 0)

is.int64(x)

## Default S3 method:
as.int64(x, ...)

## S3 method for class 'factor'
as.int64(x, ...)

## S3 method for class 'character'
as.int64(x, base = 10L, ...)

## S3 method for class 'numeric'
as.int64(x, ...)

## S3 method for class 'NULL'
as.int64(x, ...)

## S3 method for class 'int64'
format(x, ...)

## S3 method for class 'int64'
print(x, ...)

## S3 method for class 'int64'
as.character(x, base = NULL, ...)

## S3 method for class 'int64'
as.double(x, ...)

## S3 method for class 'int64'
as.integer(x, ...)

## S3 method for class 'int64'
is.na(x, ...)

## S3 method for class 'int64'
as.data.frame(x, ...)
```

```
## S3 method for class 'int64'
as.list(x, ...)

## S3 method for class 'int64'
c(...)

## S3 method for class 'int64'
is.numeric(x)

## S3 method for class 'int64'
rep(x, ...)
```

### Arguments

length	A non-negative integer specifying the desired length. Double values will be coerced to integer: supplying an argument of length other than one is an error.
x	Object to be coerced or tested
...	Further arguments passed to or from other methods.
base	Specifies the base of the number (default is the base attribute of the object).

### Details

The `int64` class stores 64-bit integers in vectors of doubles and the base as an attribute `base` of the vector for printing and conversion to character. The motivation behind this class is to give R the ability to load 64-bit integers directly, for example, to represent the commonly used 64-bit identifiers in relational and other databases.

### See Also

Ops.int64 csvread

---

Ops.int64

*Operators for the int64 class.*

---

### Description

Operators for the `int64` class: one of `+`, `-`, `==`, `!=`, `<`, `<=`, `>` or `>=`.

### Usage

```
e1 + e2
e1 - e2

## S3 method for class 'int64'
e1 + e2

## S3 method for class 'int64'
e1 - e2
```

**Arguments**

- |    |  |
|----|--|
| e1 | int64 object, character vector or numeric vector (character and numeric values are converted by <code>as.int64</code> ). |
| e2 | int64 object, character vector or numeric vector (character and numeric values are converted by <code>as.int64</code> ). |

**See Also**

int64



# Index

\*Topic **64-bit**  
    csvread, 2

\*Topic **bigint**  
    csvread, 2

\*Topic **comma-separated**  
    csvread, 2

\*Topic **csv**  
    csvread, 2

\*Topic **delimited**  
    csvread, 2

\*Topic **file**  
    csvread, 2

\*Topic **import**  
    csvread, 2

\*Topic **integer64**  
    csvread, 2

\*Topic **read.csv**  
    csvread, 2

\*Topic **text**  
    csvread, 2

+ (Ops.int64), 7

- (Ops.int64), 7

< (Ops.int64), 7

[.int64 (int64), 6

[<-.int64 (int64), 6

[.int64 (int64), 6

as.character.int64 (int64), 6

as.data.frame.int64 (int64), 6

as.double.int64 (int64), 6

as.int64 (int64), 6

as.integer.int64 (int64), 6

as.list.int64 (int64), 6

c.int64 (int64), 6

csvread, 2

csvread-package (csvread), 2

format.int64 (int64), 6

int64, 2, 3, 6

is.int64 (int64), 6

is.na.int64 (int64), 6

is.numeric.int64 (int64), 6

map.coltypes (csvread), 2

Ops.int64, 7

print.int64 (int64), 6

rep.int64 (int64), 6