

Package ‘R.huge’

January 2, 2012

Version 0.3.0

Date 2011-07-23

Title Methods for accessing huge amounts of data [DEPRECATED]

Author Henrik Bengtsson <henrikb@braju.com>

Maintainer Henrik Bengtsson <henrikb@braju.com>

Depends R (>= 2.5.0), R.oo (>= 1.8.1), R.utils (>= 1.7.5)

Description THIS PACKAGE HAS BEEN DEPRECATED. Do not start building new projects based on it. Cross-platform alternatives are the following packages: bigmemory (CRAN), ff (CRAN), BufferedMatrix (BioConductor). The main usage of it was inside the aroma.affymetrix package. (The package currently provides a class representing a matrix where the actual data is stored in a binary format on the local file system. This way the size limit of the data is set by the file system and not the memory.)

License LGPL (>= 2.1)

URL <http://www.braju.com/R/>

LazyLoad TRUE

Repository CRAN

Date/Publication 2011-07-24 05:14:06

R topics documented:

R.huge-package	2
AbstractFileArray	3
FileMatrix	6
FileVector	9

Index	13
--------------	-----------

R.huge-package

Package R.huge

Description

This package has been deprecated. Do not start building new projects based on it.

THIS PACKAGE HAS BEEN DEPRECATED. Do not start building new projects based on it. Cross-platform alternatives are the following packages: bigmemory (CRAN), ff (CRAN), Buffered-Matrix (BioConductor). The main usage of it was inside the aroma.affymetrix package. (The package currently provides a class representing a matrix where the actual data is stored in a binary format on the local file system. This way the size limit of the data is set by the file system and not the memory.)

Requirements

This package requires the **R.oo** package [1] and the **R.utils** package.

Installation and updates

To install this package, see <http://www.braju.com/R/>.

To get started

To get started, see:

1. [FileVector](#).
2. [FileMatrix](#).

How to cite this package

Please cite [1] below.

License

The releases of this package is licensed under LGPL version 2.1 or newer.

The development code of the packages is under a private licence (where applicable) and patches sent to the author fall under the latter license, but will be, if incorporated, released under the "release" license above.

References

- 1 H. Bengtsson, *The R.oo package - Object-Oriented Programming with References Using Standard R Code*, In Kurt Hornik, Friedrich Leisch and Achim Zeileis, editors, Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003), March 20-22, Vienna, Austria. <http://www.ci.tuwien.ac.at/Conferences/DSC-2003/Proceedings/>

Author(s)

Henrik Bengtsson (<http://www.braju.com/R/>)

AbstractFileArray *Class representing a persistent array stored in a file*

Description

Package: R.huge

Class AbstractFileArray**Object**

```
~~|
~~+--AbstractFileArray
```

Directly known subclasses:

[FileByteMatrix](#), [FileByteVector](#), [FileDoubleMatrix](#), [FileDoubleVector](#), [FileFloatMatrix](#), [FileFloatVector](#), [FileIntegerMatrix](#), [FileIntegerVector](#), [FileMatrix](#), [FileShortMatrix](#), [FileShortVector](#), [FileVector](#)

```
public static class AbstractFileArray
```

```
extends Object
```

Note that this is an abstract class, i.e. it is not possible to create an object of this class but only from one of its subclasses. For a vector data type, see [FileVector](#). For a matrix data type, see [FileMatrix](#).

Usage

```
AbstractFileArray(filename=NULL, path=NULL, storageMode=c("integer", "double"), bytesPerCell=1, dim=N
```

Arguments

filename	The name of the file storing the data.
path	An optional path where data should be stored.
storageMode	The storage mode() of the data elements.
bytesPerCell	The number of bytes each element (cell) takes up on the file system. If NULL, it is inferred from the storageMode argument.
dim	A numeric vector specifying the dimensions of the array.
dimnames	An optional list of dimension names.
dimOrder	The order of the dimensions.
comments	An optional character string of arbitrary length.
nbrOffFreeBytes	The number of "spare" bytes after the comments before the data section begins.

Details

The purpose of this class is to be able to work with large arrays in R without being limited by the amount of memory available. Data is kept on the file system and elements are read and written whenever queried.

Fields and Methods

Methods:

<code>as.character</code>	Returns a short string describing the file array.
<code>as.vector</code>	Returns the elements of a file array as an R vector.
<code>clone</code>	Clones a file array.
<code>close</code>	Closes a connection to the data file of the file array.
<code>delete</code>	Deletes the file array from the file system.
<code>dim</code>	Gets the dimension of the file array.
<code>dimnames</code>	Gets the dimension names of a file array.
<code>finalize</code>	Internal: Clean up when file array is deallocated from memory.
<code>flush</code>	Internal: Flushes the write buffer.
<code>getBaseline</code>	Gets the basename (filename) of the data file.
<code>getBytesPerCell</code>	Gets the number of bytes per element in a file array.
<code>getCloneNumber</code>	Gets the clone number of the file array.
<code>getComments</code>	Gets the comments for this file array.
<code>getDataOffset</code>	Gets file position of the data section in a file array.
<code>getDimensionOrder</code>	Gets the order of dimension.
<code>getExtension</code>	Gets the filename extension of the file array.
<code>getFileSize</code>	Gets the size of the file array.
<code>getName</code>	Gets the name of the file array.
<code>getPath</code>	Gets the path (directory) where the data file lives.
<code>getPathname</code>	Gets the full pathname to the data file.
<code>getsizeofComments</code>	Gets the number of bytes the comments occupies.
<code>getsizeofData</code>	Gets the size of the data section in bytes.
<code>getStorageMode</code>	Gets the storage mode of the file array.
<code>isOpen</code>	Checks whether the data file of the file array is open or not.
<code>length</code>	Gets the number of elements in a file array.
<code>open</code>	Opens a connection to the data file of the file array.
<code>readAllValues</code>	Reads all values in the file array.
<code>readContiguousValues</code>	Reads sets of contiguous values in the file array.
<code>readHeader</code>	Read the header of a file array data file.
<code>readValues</code>	Reads individual values in the file array.
<code>setComments</code>	Sets the comments for this file array.
<code>writeAllValues</code>	Writes all values to a file array.
<code>writeEmptyData</code>	Writes an empty data section to the data file of a file array.
<code>writeHeader</code>	Writes the header of a file array to file.
<code>writeHeaderComments</code>	-
<code>writeValues</code>	Writes values to a file array.

Methods inherited from Object:

\$, \$<-, [], [[<-, as.character, attach, attachLocally, clearCache, clone, detach, equals, extend, finalize, gc, getEnvironment, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, objectSize, print, registerFinalizer, save

Maximum number of elements

It is only the header that is kept in memory, not the data, and therefore the maximum length of a array that can be allocate, is limited by the amount of available space on the file system. Since element names (optional) are stored in the header, these may also be a limiting factor.

Element names

The element names are stored in the header and are currently read and written to file one by one. This may slow down the performance substantially if the dimensions are large. For optimal opening performance, avoid names.

For now, do *not* change names after file has been allocated.

File format

The file format consist of a header section and a data section. The header contains information about the file format, the length and element names of the array, as well as data type (storage `mode()`), the size of each element. The data section, which follows immediately after the header section, consists of all data elements with non-assigned elements being pre-allocated with zeros.

For more details, see the source code.

Limitations

The size of the array in bytes is limited by the maximum file size of the file system. For instance, the maximum file size on a Windows FAT32 system is 4GB (2GB?). On Windows NTFS the limit is in practice ~16TB.

Author(s)

Henrik Bengtsson (<http://www.braju.com/R/>)

References

[1] New Technology File System (NTFS), Wikipedia, 2006 <http://en.wikipedia.org/wiki/NTFS>.

FileMatrix

*Class representing a persistent matrix stored in a file***Description**

Package: R.huge

Class FileMatrix

Object

```

~~|
~~+--AbstractFileArray
~~~~~|
~~~~~+--FileMatrix

```

Directly known subclasses:[FileByteMatrix](#), [FileDoubleMatrix](#), [FileFloatMatrix](#), [FileIntegerMatrix](#), [FileShortMatrix](#)

```

public static class FileMatrix
  extends AbstractFileArray

```

Usage

```
FileMatrix(..., nrow=NULL, ncol=NULL, rownames=NULL, colnames=NULL, byrow=FALSE)
```

Arguments

```

...           Arguments passed to AbstractFileArray.
nrow, ncol    The number of rows and columns of the matrix.
rownames, colnames
              Optional row and column names.
byrow        If TRUE, data are stored row by row, otherwise column by column.

```

Details

The purpose of this class is to be able to work with large matrices in R without being limited by the amount of memory available. Matrices are kept on the file system and elements are read and written whenever queried. The purpose of the class is *not* to provide methods for full matrix operations, but instead to be able to work with subsets of the matrix at each time.

For more details, [AbstractFileArray](#).

Fields and Methods**Methods:**

[-
[<-	-
as.character	Returns a short string describing the file matrix.
as.matrix	Returns the elements of a file matrix as an R matrix.
colnames	Gets the column names of a file matrix.
getByRow	Checks if elements are stored row by row or not.
getColumnOffset	-
getMatrixIndicies	-
getOffset	-
getRowOffset	-
ncol	Gets the number of columns of the matrix.
nrow	Gets the number of rows of the matrix.
readFullMatrix	-
readValues	-
rowMeans	Calculates the means for each row.
rownames	Gets the row names of a file matrix.
rowSums	Calculates the sum for each row.
writeValues	-

Methods inherited from AbstractFileArray:

as.character, as.vector, clone, close, delete, dim, dimnames, finalize, flush, getBaseline, getBytesPerCell, getCloneNumber, getComments, getDataOffset, getDimensionOrder, getExtension, getFileSize, getName, getPath, getPathname, getSizeOfComments, getSizeOfData, getStorageMode, isOpen, length, open, readAllValues, readContiguousValues, readHeader, readValues, setComments, writeAllValues, writeEmptyData, writeHeader, writeHeaderComments, writeValues

Methods inherited from Object:

\$. \$<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clone, detach, equals, extend, finalize, gc, getEnvironment, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, objectSize, print, registerFinalizer, save

Column by column or row by row?

If the matrix elements are to be accessed more often along rows, store data row by row, otherwise column by column.

Supported data types

The following subclasses implement support for various data types:

- FileByteMatrix (1 byte per element),
- FileShortMatrix (2 bytes per element),
- FileIntegerMatrix (4 bytes per element),
- FileFloatMatrix (4 bytes per element), and
- FileDoubleMatrix (8 bytes per element).

Author(s)

Henrik Bengtsson (<http://www.braju.com/R/>)

Examples

```

verbose <- Arguments$getVerbose(TRUE);

pathname <- "example.Rmatrix";
if (isFile(pathname)) {
  file.remove(pathname);
  if (isFile(pathname)) {
    stop("File not deleted: ", pathname);
  }
}

# -----
# Create a new file matrix
# -----
verbose && enter(verbose, "Creating new matrix");
# The dimensions of the matrix
nrow <- 20;
ncol <- 5;
X <- FileByteMatrix(pathname, nrow=nrow, ncol=ncol, byrow=TRUE);
verbose && exit(verbose);

verbose && enter(verbose, "Filling it with data");
rows <- c(1:4,7:10);
cols <- c(1);
x <- 1:length(rows);
writeValues(X, rows=rows, cols=cols, values=x);
verbose && exit(verbose);

verbose && enter(verbose, "Getting data again");
y <- readValues(X, rows=rows, cols=cols);
verbose && exit(verbose);
stopifnot(all.equal(x,y));

verbose && enter(verbose, "Setting data using [i,j]");
i <- c(20:18, 13:15); j <- c(3:2, 4:5);
n <- length(i) * length(j);
values <- 1:n;
X[i,j] <- values;
verbose && enter(verbose, "Validating");
print(X);
print(X[])
print(X[i,j]);
stopifnot(all.equal(as.vector(X[i,j]), values))
verbose && exit(verbose);
verbose && exit(verbose);

# -----

```

```

# Open an already existing file matrix
# -----
verbose && enter(verbose, "Getting existing matrix");
Y <- FileByteMatrix(pathname);
verbose && exit(verbose);

print(Y[])
Y[5,1] <- 55;
print(Y[])
print(X[]) # Note, X and Y refers to the same instance

# -----
# Clone a matrix
# -----
Z <- clone(X)
Z[5,1] <- 66;
print(Z[])
print(Y[])

# Remove clone again
delete(Z);

# -----
# Close all matrices
# -----
close(X);
close(Y);

# Remove original matrix too
delete(X);

gc();

```

FileVector

Class representing a persistent vector stored on file

Description

Package: R.huge
Class FileVector

Object

```

~~|
~~+--AbstractFileArray
~~~~~|
~~~~~+--FileVector

```

Directly known subclasses:

[FileByteVector](#), [FileDoubleVector](#), [FileFloatVector](#), [FileIntegerVector](#), [FileShortVector](#)

```
public static class FileVector
  extends AbstractFileArray
```

Usage

```
FileVector(..., length=NULL, names=NULL)
```

Arguments

...	Arguments passed to AbstractFileArray .
length	The number of elements in the vector.
names	Optional element names.

Details

The purpose of this class is to be able to work with large vectors in R without being limited by the amount of memory available. Data is kept on the file system and elements are read and written whenever queried.

For more details, [AbstractFileArray](#).

Fields and Methods**Methods:**

[-
[<-	-
names	Gets the element names of a file vector.

Methods inherited from AbstractFileArray:

[as.character](#), [as.vector](#), [clone](#), [close](#), [delete](#), [dim](#), [dimnames](#), [finalize](#), [flush](#), [getBasename](#), [getBytesPerCell](#), [getCloneNumber](#), [getComments](#), [getDataOffset](#), [getDimensionOrder](#), [getExtension](#), [getFileSize](#), [getName](#), [getPath](#), [getPathname](#), [getSizeOfComments](#), [getSizeOfData](#), [getStorageMode](#), [isOpen](#), [length](#), [open](#), [readAllValues](#), [readContiguousValues](#), [readHeader](#), [readValues](#), [setComments](#), [writeAllValues](#), [writeEmptyData](#), [writeHeader](#), [writeHeaderComments](#), [writeValues](#)

Methods inherited from Object:

[\\$](#), [\\$<-](#), [\[\[](#), [\[\[<-](#), [as.character](#), [attach](#), [attachLocally](#), [clearCache](#), [clone](#), [detach](#), [equals](#), [extend](#), [finalize](#), [gc](#), [getEnvironment](#), [getFields](#), [getInstantiationTime](#), [getStaticInstance](#), [hasField](#), [hashCode](#), [ll](#), [load](#), [objectSize](#), [print](#), [registerFinalizer](#), [save](#)

Supported data types

The following subclasses implement support for various data types:

- FileByteVector (1 byte per element),
- FileShortVector (2 bytes per element),
- FileIntegerVector (4 bytes per element),
- FileFloatVector (4 bytes per element), and
- FileDoubleVector (8 bytes per element).

Author(s)

Henrik Bengtsson (<http://www.braju.com/R/>)

Examples

```

verbose <- Arguments$getVerbose(TRUE);

pathname <- "example.Rvector";
if (isFile(pathname)) {
  file.remove(pathname);
  if (isFile(pathname)) {
    stop("File not deleted: ", pathname);
  }
}

# -----
# Create a new file vector
# -----
verbose && enter(verbose, "Creating new vector");
# The length of the vector
length <- 1e6;
X <- FileDoubleVector(pathname, length=length);
verbose && exit(verbose);
print(X);

verbose && enter(verbose, "Filling it with data");
idxs <- c(1:4,7:10);
x <- 1:length(idxs);
writeValues(X, indices=idxs, values=x);
verbose && exit(verbose);

verbose && enter(verbose, "Getting data again");
y <- readValues(X, indices=idxs);
verbose && exit(verbose);
stopifnot(all.equal(x,y));

verbose && enter(verbose, "Getting and setting data using [i,j]");
print(X[1:20])
i <- 13:15;
X[i] <- 99:98;
print(X[1:20])
verbose && exit(verbose);

delete(X);

```

```
rm(X);
```

```
gc();
```

Index

*Topic **classes**

- AbstractFileArray, 3
- FileMatrix, 6
- FileVector, 9

*Topic **package**

- R.huge-package, 2

AbstractFileArray, 3, 6, 9, 10

as.character, 4, 7

as.matrix, 7

as.vector, 4

character, 3

clone, 4

close, 4

colnames, 7

delete, 4

dim, 4

dimnames, 4

FileByteMatrix, 3, 6

FileByteMatrix (FileMatrix), 6

FileByteVector, 3, 10

FileByteVector (FileVector), 9

FileDoubleMatrix, 3, 6

FileDoubleMatrix (FileMatrix), 6

FileDoubleVector, 3, 10

FileDoubleVector (FileVector), 9

FileFloatMatrix, 3, 6

FileFloatMatrix (FileMatrix), 6

FileFloatVector, 3, 10

FileFloatVector (FileVector), 9

FileIntegerMatrix, 3, 6

FileIntegerMatrix (FileMatrix), 6

FileIntegerVector, 3, 10

FileIntegerVector (FileVector), 9

FileMatrix, 2, 3, 6

FileShortMatrix, 3, 6

FileShortMatrix (FileMatrix), 6

FileShortVector, 3, 10

FileShortVector (FileVector), 9

FileVector, 2, 3, 9

finalize, 4

flush, 4

getBasename, 4

getByRow, 7

getBytesPerCell, 4

getCloneNumber, 4

getComments, 4

getDataOffset, 4

getDimensionOrder, 4

getExtension, 4

getFileSize, 4

getName, 4

getPath, 4

getPathname, 4

getSizesOfComments, 4

getSizesOfData, 4

getStorageMode, 4

isOpen, 4

length, 4

list, 3

mode, 3, 5

names, 10

ncol, 7

nrow, 7

numeric, 3

Object, 3, 6, 9

open, 4

R.huge (R.huge-package), 2

R.huge-package, 2

readAllValues, 4

readContiguousValues, 4

readHeader, 4
readValues, 4
rowMeans, 7
rownames, 7
rowSums, 7

setComments, 4

TRUE, 6

vector, 3

writeAllValues, 4
writeEmptyData, 4
writeHeader, 4
writeValues, 4