Package 'LaF'

July 21, 2025

Type Package

Title Fast Access to Large ASCII Files

Version 0.8.6

Date 2024-12-13

Description Methods for fast access to large ASCII files. Currently the following file formats are supported: comma separated format (CSV) and fixed width format. It is assumed that the files are too large to fit into memory, although the package can also be used to efficiently access files that do fit into memory. Methods are provided to access and process files blockwise. Furthermore, an opened file can be accessed as one would an ordinary data.frame. The LaF vignette gives an overview of the functionality provided.

URL https://github.com/djvanderlaan/LaF

License GPL-3

LazyLoad yes

Depends methods, utils

Suggests testthat, yaml

LinkingTo Rcpp

Imports Rcpp (>= 0.11.1)

Collate 'generics.R' 'laf.R' 'laf_column.R' 'meta.R' 'open.R' 'read_dm_blaise.R' 'stats.R' 'textutils.R' 'types.R' 'utility.R'

RoxygenNote 7.3.1

Encoding UTF-8

NeedsCompilation yes

Author Jan van der Laan [aut, cre] (ORCID: <https://orcid.org/0000-0002-0693-1514>)

Maintainer Jan van der Laan <r@eoos.dds.nl>

Repository CRAN

Date/Publication 2024-12-13 16:50:02 UTC

Contents

`begin` *Go to the beginning of the file*

Description

Sets the file pointer to the beginning of the file. The next call to `next_block` returns the first lines of the file. This method is usually used in combination with `next_block`.

Usage

```
begin(x, ...)

## S4 method for signature 'laf'
begin(x, ...)
```

Arguments

x	an object the supports the begin method, such as an laf object.
...	passed to other methods.

---

close,laf-method *Close the connection to the Large File*

Description

Close the connection to the Large File

Usage

```
## S4 method for signature 'laf'
close(con, ...)
```

Arguments

con	a "laf" object that can be closed.
...	unused.

---

colsum *Calculate simple statistics of column*

Description

Methods for calculating simple statistics of columns of a file: mean, sum, standard deviation, range (min and max), and number of missing values.

Usage

```
colsum(x, ...)

## S4 method for signature 'laf'
colsum(x, columns, na.rm = TRUE, ...)

## S4 method for signature 'laf_column'
colsum(x, na.rm = TRUE, ...)

colmean(x, ...)

## S4 method for signature 'laf'
colmean(x, columns, na.rm = TRUE, ...)
```

```
## S4 method for signature 'laf_column'
colmean(x, na.rm = TRUE, ...)

colfreq(x, ...)

## S4 method for signature 'laf'
colfreq(x, columns, useNA = c("ifany", "always", "no"), ...)

## S4 method for signature 'laf_column'
colfreq(x, na.rm = TRUE, ...)

colrange(x, ...)

## S4 method for signature 'laf'
colrange(x, columns, na.rm = TRUE, ...)

## S4 method for signature 'laf_column'
colrange(x, na.rm = TRUE, ...)

colnmissing(x, ...)

## S4 method for signature 'laf'
colnmissing(x, columns, na.rm = TRUE, ...)

## S4 method for signature 'laf_column'
colnmissing(x, na.rm = TRUE, ...)
```

Arguments

x	an object of type laf or laf_column.
...	Currently ignored.
columns	a numeric vector with the columns for which the statistics should be calculated.
na.rm	whether or not to ignore missing values. By default missing values are ignored.
useNA	method with which to treat missing values: "ifany" adds a field containing the number of missing values if there are any; "always" will always add a field with the number of missing values even when there are none; "none" will never add a field containing the number of missing values.

---

current_line *Get the current line in the file*

Description

Get the current line in the file

Usage

```
current_line(x)

## S4 method for signature 'laf'
current_line(x)
```

Arguments

x	an object the supports the current_line method, such as an laf object. Returns the next line that will be read by next_block. The current line can be set by the method goto.

---

detect_dm_csv *Automatically detect data models for CSV-files*

Description

Automatically detect data models for CSV-files. Opening of files using the data models can be done using laf_open.

Usage

```
detect_dm_csv(
  filename,
  sep = ",",
  dec = ".",
  header = FALSE,
  nrows = 1000,
  nlines = NULL,
  sample = FALSE,
  stringsAsFactors = TRUE,
  factor_fraction = 0.4,
  ...
)
```

Arguments

filename	character containing the filename of the csv-file.
sep	character vector containing the separator used in the file.
dec	the character used for decimal points.
header	does the first line in the file contain the column names.
nrows	the number of lines that should be read in to detect the column types. The more lines the more likely that the correct types are detected.
nlines	(only needed when the sample option is used) the expected number of lines in the file. If not specified the number of lines in the file is first calculated.

`sample` by default the first `nrows` lines are read in for determining the column types. When sample is used random lines from the file are used. This is more robust, but takes longer.

`stringsAsFactors` passed on to `read.table`. Set to `FALSE` to read all text columns as character. In that case `factor_fraction` is ignored.

`factor_fraction` the fraction of unique string in a column below which the column is converted to a factor/categorical. For more information see details.

`...` additional arguments are passed on to `read.table`. However, be careful with using these as some of these arguments are not supported by `laf_open_csv`.

Details

The argument `factor_fraction` determines the fraction of unique strings below which the column is converted to factor/categorical. If all column need to be converted to character a value larger than one can be used. A value smaller than zero will ensure that all columns will be converted to categorical. Note that LaF stores the levels of a categorical in memory. Therefore, for categorical columns with a very large number of (almost) unique levels can cause memory problems.

Value

`read_dm` returns a data model which can be used by `laf_open`. The data model can be written to file using `write_dm`.

See Also

See `write_dm` to write the data model to file. The data models can be used to open a file using `laf_open`.

Examples

```
# Create temporary filename
tmpcsv  <- tempfile(fileext="csv")

# Generate test data
ntest <- 10
column_types <- c("integer", "integer", "double", "string")
testdata <- data.frame(
    a = 1:ntest,
    b = sample(1:2, ntest, replace=TRUE),
    c = round(runif(ntest), 13),
    d = sample(c("jan", "pier", "tjores", "corneel"), ntest, replace=TRUE),
    stringsAsFactors = FALSE
    )
# Write test data to csv file
write.table(testdata, file=tmpcsv, row.names=FALSE, col.names=TRUE, sep=',')

# Detect data model
model <- detect_dm_csv(tmpcsv, header=TRUE)
```

```
# Create LaF-object
laf <- laf_open(model)

# Cleanup
file.remove(tmpcsv)
```

`determine_nlines` *Determine number of lines in a text file*

Description

Determine number of lines in a text file

Usage

```
determine_nlines(filename)
```

Arguments

`filename`	character containing the filename of the file of which the lines are to be counted.

Details

The routine counts the number of line endings. If the last line does not end in a line ending, but does contain character, this line is also counted.

The file size is not limited by the amount of memory in the computer.

Value

Returns the number of lines in the file.

See Also

See `readLines` to read in all lines a text file; `get_lines` and `sample_lines` can be used to read in specified, or random lines.

Examples

```
# Create temporary filename
tmpcsv  <- tempfile(fileext="csv")

# Generate file
writeLines(letters[1:20], con=tmpcsv)

# Count the lines
determine_nlines(tmpcsv)
```

```
# Cleanup
file.remove(tmpcsv)
```

`get_lines` *Read in specified lines from a text file*

Description

Read in specified lines from a text file

Usage

```
get_lines(filename, line_numbers)
```

Arguments

`filename`	character containing the filename of the file from which the lines should be read.
`line_numbers`	A vector containing the lines that should be read.

Details

Line numbers larger than the number of lines in the file are ignored. Missing values are returned for these.

Value

Returns a character vector with the specified lines.

See Also

See `readLines` to read in all lines a text file; `sample_lines` can be used to read in random lines.

Examples

```
# Create temporary filename
tmpcsv  <- tempfile(fileext="csv")

writeLines(letters[1:20], con=tmpcsv)
get_lines(tmpcsv, c(1, 10))

# Cleanup
file.remove(tmpcsv)
```

goto *Go to specified line in the file*

Description

Sets the current line to the line number specified. The next call to `next_block` will return the data on the specified line in the first row. The number of the current line can be obtained using `current_line`.

Usage

```
goto(x, i, ...)

## S4 method for signature 'laf,numeric'
goto(x, i, ...)
```

Arguments

`x`	an object the supports the `goto` method, such as an `laf` object.
`i`	the line number .
`...`	additional parameters passed to other methods.

laf-class *Large File object*

Description

A Large File object. This is a reference to a dataset on disk. The data itself is not read into memory (yet). This can be done by the methods for blockwise processing or by indexing the object as a data.frame. The code has been optimised for fast access.

Objects from the Class

Objects can be created by opening a file using one of the methods `laf_open_csv` or `laf_open_fwf`. These create a reference to either a CSV file or a fixed width file. The data in these files can either be accessed using blockwise operations using the methods `begin`, `next_block` and `goto`. Or by indexing the laf object as you would a data.frame. In the following example a CSV file is opened and its first column (of type integer) is read into memory:

```
laf <- laf_open_csv("file.csv", column_types=c("integer", "double"))
data <- laf[ , 1]
```

laf_column-class *Column of a Large File Object*

Description

Representation of a column in a Large File object. This class itself is a subclass of the class `laf`. In principle all methods that can be used with a `laf` object can also be used with a `laf_column` object except the the `column` or `columns` arguments of these methods are not needed.

Objects from the Class

Object of this class are usually created by using the `$` operator on `laf` objects.

laf_open *Create a connection to a file using a data model.*

Description

Uses a data model to create a connection to a file. The data model contains all the information needed to open the file (column types, column widths, etc.).

Usage

```
laf_open(model, ...)
```

Arguments

`model`	a data model, such as one returned by read_dm or detect_dm_csv.
`...`	additional arguments can be used to overwrite the values specified by the data model. These are listed in the argument documentation for `laf_open_csv` and `laf_open_fwf`, e.g. see `ignore_failed_conversion`.

Details

Depending on the field 'type' `laf_open` uses `laf_open_csv` and `laf_open_fwf` to open the file. The data model should contain all information needed by these routines to open the file.

Value

Object of type `laf`. Values can be extracted from this object using indexing, and methods such as `read_lines`, `next_block`.

See Also

See `read_dm` and `detect_dm_csv` for ways of creating data models.

Examples

```
# Create some temporary files
tmpcsv  <- tempfile(fileext="csv")
tmp2csv <- tempfile(fileext="csv")
tmpyaml <- tempfile(fileext="yaml")

# Generate test data
ntest <- 10
column_types <- c("integer", "integer", "double", "string")
testdata <- data.frame(
    a = 1:ntest,
    b = sample(1:2, ntest, replace=TRUE),
    c = round(runif(ntest), 13),
    d = sample(c("jan", "pier", "tjores", "corneel"), ntest, replace=TRUE)
    )
# Write test data to csv file
write.table(testdata, file=tmpcsv, row.names=FALSE, col.names=FALSE, sep=',')

# Create LaF-object
laf <- laf_open_csv(tmpcsv, column_types=column_types)

# Write data model to file
write_dm(laf, tmpyaml)

# Read data model and open file
laf <- laf_open(read_dm(tmpyaml))

# Write test data to second csv file
write.table(testdata, file=tmp2csv, row.names=FALSE, col.names=FALSE, sep=',')

# Read data model and open second file, demonstrating the use of the optional
# arguments to laf_open
laf2 <- laf_open(read_dm(tmpyaml), filename=tmp2csv)

# Cleanup
file.remove(tmpcsv)
file.remove(tmp2csv)
file.remove(tmpyaml)
```

`laf_open_csv` *Create a connection to a comma separated value (CSV) file.*

Description

A connection to the file filename is created. Column types have to be specified. These are not determined automatically as for example read.csv does. This has been done to increase speed.

Usage

```
laf_open_csv(
  filename,
  column_types,
  column_names = paste("V", seq_len(length(column_types)), sep = ""),
  sep = ",",
  dec = ".",
  trim = FALSE,
  skip = 0,
  ignore_failed_conversion = FALSE
)
```

Arguments

filename	character containing the filename of the CSV-file
column_types	character vector containing the types of data in each of the columns. Valid types are: double, integer, categorical and string.
column_names	optional character vector containing the names of the columns.
sep	optional character specifying the field separator used in the file.
dec	optional character specifying the decimal mark.
trim	optional logical specifying whether or not white space at the end of factor levels or character strings should be trimmed.
skip	optional numeric specifying the number of lines at the beginning of the file that should be skipped.
ignore_failed_conversion	ignore (set to NA) fields that could not be converted.

Details

After the connection is created data can be extracted using indexing (as in a normal data.frame) or methods such as read_lines and next_block can be used to read in blocks. For processing the file in blocks the convenience function process_blocks can be used.

The CSV-file should not contain headers. Use the skip option to skip any headers.

In case of an incomplete line (at line with less columns than it should have): when the line is completely empty the reader stops at that point and considers that as the end of the file. In other cases a warning is issued and the remaining columns are considered empty. For character columns this results in an empty string for numeric columns a NA.

Value

Object of type laf. Values can be extracted from this object using indexing, and methods such as read_lines, next_block.

See Also

See read.csv for conventional access of CSV files. And detect_dm_csv to automatically determine the column types.

Examples

```
# Create temporary filename
tmpcsv  <- tempfile(fileext="csv")

# Generate test data
ntest <- 10
column_types <- c("integer", "integer", "double", "string")
testdata <- data.frame(
    a = 1:ntest,
    b = sample(1:2, ntest, replace=TRUE),
    c = round(runif(ntest), 13),
    d = sample(c("jan", "pier", "tjores", "corneel"), ntest, replace=TRUE)
    )
# Write test data to csv file
write.table(testdata, file=tmpcsv, row.names=FALSE, col.names=FALSE, sep=',')

# Create LaF-object
laf <- laf_open_csv(tmpcsv, column_types=column_types)

# Read from file using indexing
first_column <- laf[ , 1]
first_row    <- laf[1, ]

# Read from file using blockwise operators
begin(laf)
first_block <- next_block(laf, nrows=2)
second_block <- next_block(laf, nrows=2)

# Cleanup
file.remove(tmpcsv)
```

`laf_open_fwf` *Create a connection to a fixed width file.*

Description

A connection to the file filename is created. Column types have to be specified. These are not determined automatically as for example read.fwf does. This has been done to increase speed.

Usage

```
laf_open_fwf(
  filename,
  column_types,
  column_widths,
  column_names = paste("V", seq_len(length(column_types)), sep = ""),
  dec = ".",
  trim = TRUE,
```

```
  ignore_failed_conversion = FALSE
)
```

Arguments

filename	character containing the filename of the fixed width file.
column_types	character vector containing the types of data in each of the columns. Valid types are: double, integer, categorical and string.
column_widths	numeric vector containing the width in number of character of each of the columns.
column_names	optional character vector containing the names of the columns.
dec	optional character specifying the decimal mark.
trim	optional logical specifying whether or not whitespace at the end of factor levels or character strings should be trimmed.
ignore_failed_conversion	ignore (set to NA) fields that could not be converted.

Details

After the connection is created data can be extracted using indexing (as in a normal data.frame) or methods such as read_lines and next_block can be used to read in blocks. For processing the file in blocks the (faster) convenience function process_blocks can be used.

Only use `ignore_failed_conversion` when you are sure that the column specification is correct. Otherwise, this option can hide an incorrect specification.

Value

Object of type `laf`. Values can be extracted from this object using indexing, and methods such as `read_lines`, `next_block`.

See Also

See `read.fwf` for conventional access of fixed width files.

---

levels,laf-method *Get and change the levels of the column in a Large File object*

Description

Get and change the levels of the column in a Large File object

Usage

```
## S4 method for signature 'laf'
levels(x)

## S4 replacement method for signature 'laf'
levels(x) <- value

## S4 method for signature 'laf_column'
levels(x)

## S4 replacement method for signature 'laf_column'
levels(x) <- value
```

Arguments

x	a "laf" object.
value	a list with the levels for each column.

---

names,laf-method *Get and set the names of the columns in a Large File object*

Description

Get and set the names of the columns in a Large File object

Usage

```
## S4 method for signature 'laf'
names(x)

## S4 replacement method for signature 'laf'
names(x) <- value
```

Arguments

x	a "laf" object.
value	a character vector with the new column names

ncol,laf-method *Get the number of columns in a Large File object*

Description

Get the number of columns in a Large File object

Usage

```
## S4 method for signature 'laf'
ncol(x)
```

Arguments

x	a "laf" object.

next_block *Read the next block of data from a file.*

Description

Read the next block of data from a file.

Usage

```
next_block(x, ...)

## S4 method for signature 'laf'
next_block(x, columns = 1:ncol(x), nrows = 5000, ...)

## S4 method for signature 'laf_column'
next_block(x, nrows = 5000, ...)
```

Arguments

x	an object the supports the next_block method, such as an laf object.
...	passed to other methods.
	Reads the next block of lines from a file. The method returns a data.frame. The first line in the data.frame is the line corresponding to the current line in the file. When the end of the file is reached a data.frame with zero rows is returned. This can be used to check whether the end of the file is reached.
columns	an integer vector with the columns that should be read in.
nrows	the (maximum) number of rows to read in one block

nrow,laf-method *Get the number of rows in a Large File object*

Description

Get the number of rows in a Large File object

Usage

```
## S4 method for signature 'laf'
nrow(x)
```

Arguments

x	a "laf" object.

process_blocks *Blockwise processing of file*

Description

Reads the specified file block by block and feeds each block to the specified function.

Usage

```
process_blocks(x, fun, ...)

## S4 method for signature 'laf'
process_blocks(
  x,
  fun,
  columns = 1:ncol(x),
  nrows = 5000,
  allow_interupt = FALSE,
  progress = FALSE,
  ...
)
```

Arguments

x	an object the supports the process_blocks method, such as an laf object.
fun	a function to apply to each block (see details).
...	additional parameters are passed on to fun.
columns	an integer vector with the columns that should be read in.

nrows	the (maximum) number of rows to read in one block
allow_interupt	when TRUE the function fun is expected to return a list. The second element is the result of the function. The first element should be a logical value indication whether process_blocks should continue (FALSE) or stop (TRUE). When interrupted the function is not called a last time with an empty data.frame to finalize the result.
progress	show a progress bar. Note that this triggers a calculation of the number of lines in the file which for CSV files can take some time. When numeric code is used as the style of the progress bar (see txtProgressBar).

Details

The function should accept as the first argument the next block of data. When the end of the file is reached this is an empty (zero row) data.frame. As the second argument the function should accept the output of the previous call to the function. The first time the function is called the second argument has the value NULL.

---

read_dm *Read and write data models for LaF*

Description

Using these routines data models can be written and read. These data models can be used to create LaF object without the need to specify all arguments (column names, column types etc.). Opening of files using the data models can be done using laf_open.

Usage

```
read_dm(modelfile, ...)

write_dm(model, modelfile)
```

Arguments

modelfile	character containing the filename of the file the model is to be written to/read from.
...	additional arguments are added to the data model or, when they are also present in the file are used to overwrite the values specified in the file.
model	a data model or an object of type laf. See details for more information.

Details

A data model is a list containing information which open routine should be used (e.g. laf_open_csv or laf_open_fwf), and the arguments needed for these routines. Required elements are 'type', which can (currently) be 'csv', or 'fwf', and 'columns', which should be a data.frame containing

at least the columns 'name' and 'type', and for fwf 'width'. These columns correspond to the arguments `column_names`, `column_types` and `column_widths` respectively. Other arguments of the `laf_open_*` routines can be specified as additional elements of the list.

`write_dm` can also be used to write a data model that is created from an object of type `laf`. This is probably one of the easiest ways to create a data model.

The data model is stored in a text file in YAML format which is a format in which data structures can be stored in a readable and editable format.

Value

`read_dm` returns a data model which can be used by `laf_open`.

See Also

See `detect_dm_csv` for a routine which can automatically create a data model from a CSV-file. The data models can be used to open a file using `laf_open`.

Examples

```
# Create some temporary files
tmpcsv  <- tempfile(fileext="csv")
tmp2csv <- tempfile(fileext="csv")
tmpyaml <- tempfile(fileext="yaml")

# Generate test data
ntest <- 10
column_types <- c("integer", "integer", "double", "string")
testdata <- data.frame(
    a = 1:ntest,
    b = sample(1:2, ntest, replace=TRUE),
    c = round(runif(ntest), 13),
    d = sample(c("jan", "pier", "tjores", "corneel"), ntest, replace=TRUE)
    )
# Write test data to csv file
write.table(testdata, file=tmpcsv, row.names=FALSE, col.names=FALSE, sep=',')

# Create LaF-object
laf <- laf_open_csv(tmpcsv, column_types=column_types)

# Write data model to stdout() (screen)
write_dm(laf, stdout())

# Write data model to file
write_dm(laf, tmpyaml)

# Read data model and open file
laf2 <- laf_open(read_dm(tmpyaml))

# Write test data to second csv file
write.table(testdata, file=tmp2csv, row.names=FALSE, col.names=FALSE, sep=',')
```

```
# Read data model and open seconde file, demonstrating the use of the optional
# arguments to read_dm
laf2 <- laf_open(read_dm(tmpyaml, filename=tmp2csv))

# Cleanup
file.remove(tmpcsv)
file.remove(tmp2csv)
file.remove(tmpyaml)
```

`read_dm_blaise` *Read in Blaise data models*

Description

Read in Blaise data models

Usage

```
read_dm_blaise(filename, datafilename = NA, encoding = "latin1")
```

Arguments

`filename`	the filename of the file containing the data model.
`datafilename`	the filename of the data file to which the data model belongs.
`encoding`	the encoding used in the file. See `readLines`.

Details

The function reads the data model from file and returns a list that can be used by `laf_open` to open the file for reading. Only a subset of the most common features found in Blaise files are supported. If part of the data model can not be parsed a warning is given.

Value

Returns a data model (which is a list containing all the relevant information to open a file using `laf_open`. When the file contains more than one data model a list of data models is returned and a warning issued.

See Also

See `write_dm` to write the data model to file. The data models can be used to open a file using `laf_open`.

Examples

```
# Create some temporary files
tmpdat <- tempfile(fileext="dat")
tmpbla <- tempfile(fileext="bla")

# Generate test data
lines <- c(
    " 1M 1.45Rotterdam ",
    " 2F12.00Amsterdam ",
    " 3  .22 Berlin    ",
    "  M22   Paris     ",
    " 4F12345London    ",
    " 5M     Copenhagen",
    " 6M-12.1          ",
    " 7F   -1Oslo      ")
writeLines(lines, con=tmpdat)

# Create a file containing the data model
writeLines(c(
    "DATAMODEL test",
    "FIELDS",
    "  id     : INTEGER[2]",
    "  gender : STRING[1]",
    "  x      : REAL[5] {comment}",
    "  city   : STRING[10]",
    "ENDMODEL"), con=tmpbla)
model <- read_dm_blaise(tmpbla, datafilename=tmpdat)
laf <- laf_open(model)

# Cleanup
file.remove(tmpbla)
file.remove(tmpdat)
```

`read_lines` *Read lines from the file*

Description

Reads the specified lines and columns from the data file.

Usage

```
read_lines(x, ...)

## S4 method for signature 'laf'
read_lines(x, rows, columns = 1:ncol(x), ...)

## S4 method for signature 'laf_column'
read_lines(x, rows, columns = 1:ncol(x), ...)
```

Arguments

x	an object the supports the read_lines method, such as an laf object.
...	passed on to other methods.
rows	a numeric vector with the rows that should be read from the file.
columns	an integer vector with the columns that should be read in.

Details

Note that when scanning through the complete file next_block is much faster. Also note that random file access can be slow (and is always much slower than sequential file access), especially for certain file types such as comma separated. Reading is generally faster when the lines that should be read are sorted.

---

sample_lines *Read in random lines from a text file*

Description

Read in random lines from a text file

Usage

```
sample_lines(filename, n, nlines = NULL)
```

Arguments

filename	character containing the filename of the file from which the lines should be read.
n	The number of lines that should be sampled from the file.
nlines	The total number of lines in the file. If not specified or NULL the number of lines is first determined using determine_nlines.

Details

When nlines is not specified, the total number of lines is first determined. This can take quite some time. Therefore, specifying the number of lines can cause a significant speed up. It can also be used to sample lines from the first nlines line by specifying a value for nlines that is smaller than the number of lines in the file.

Value

Returns a character vector with the sampled lines.

See Also

See readLines to read in all lines a text file; get_lines can be used to read in specified lines.

Examples

```
# Create temporary filename
tmpcsv  <- tempfile(fileext="csv")

writeLines(letters[1:20], con=tmpcsv)
sample_lines(tmpcsv, 10)

# Cleanup
file.remove(tmpcsv)
```

---

show,laf-method *Print the Large File object to screen*

Description

Print the Large File object to screen

Print a column of a Large File object to screen

Usage

```
## S4 method for signature 'laf'
show(object)

## S4 method for signature 'laf_column'
show(object)
```

Arguments

object	the object to print to screen.

[,laf-method *Read records from a large file object into R*

Description

When a connection is opened to a "laf" object; this object can then be indexed roughly as one would a data.frame.

Usage

```
## S4 method for signature 'laf'
x[i, j, drop]

## S4 method for signature 'laf_column'
x[i, j, drop]
```

Arguments

x	an object of type "laf" or "laf_column".
i	an logical or numeric vector with indices. The rows which should be selected.
j	a numeric vector with the columns to select.
drop	a logical indicating whether or not to convert the result to a vector when only one column is selected. As in when indexing a data.frame.

[[,laf-method *Select a column from a LaF object*

Description

Selecting columns from an laf object works as it does for a data.frame.

Usage

```
## S4 method for signature 'laf'
x[[i]]

## S4 method for signature 'laf'
x$name
```

Arguments

x	an object of type laf
i	index of column to select. This should be a numeric or character vector.
name	the name of the column to select.

Value

Returns an object of type laf_column. This object behaves almost the same as an laf object except that is it no longer necessary (or possible) to specify which column should be used for functions that require this.

Index