Package 'adfExplorer'

March 5, 2018

Type Package

Title Import from and Export to Amiga Disk Files

Version 0.1.4

Date 2018-03-05

Author Pepijn de Vries [aut, cre, dtc]

Maintainer Pepijn de Vries <pepijn.devries@outlook.com>

Description Amiga Disk Files (ADF) are virtual representations of 3.5 inch floppy disks for the Commodore Amiga. Most disk drives from other systems (including modern drives) are not able to read these disks. To be able to emulate this system, the ADF format was created. This package enables you to read ADF files and import and export files from and to such virtual DOS-formatted disks.

Depends R (>= 2.10)

Imports methods

License GPL-3

LazyData True

RoxygenNote 6.0.1

Suggests knitr, rmarkdown, ProTrackR

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2018-03-05 21:03:54 UTC

R topics documented:

`adf.disk.name` *Get or set the disk name of an amigaDisk object*

Description

Get or set the disk name of an `amigaDisk` object.

Usage

```
## S4 method for signature 'amigaDisk'
adf.disk.name(x)

## S4 replacement method for signature 'amigaDisk,character'
adf.disk.name(x) <- value
```

Arguments

`x`	An `amigaDisk` object for which the disk name needs to be obtained or changed.
`value`	A `character` representation with which the disk's name needs to be replaced. Disk name needs to be between 1 and 30 characters long and are not allowed to contain a colon or forward slash.

Details

DOS-formatted disks (is.amigaDOS) store their disk name on the socalled root block of the disk. This method allows you to obtain the disk's name or change it (when it is DOS-formatted).

Value

Returns A character representation of the disk's name.

Author(s)

Pepijn de Vries

Examples

```
## Not run:
data(adf.example)

## get the disk name:
adf.disk.name(adf.example)

## change it if you don't like it:
adf.disk.name(adf.example) <- "MyDisk"

## confirm that it has changed:
adf.disk.name(adf.example)

## End(Not run)
```

---

adf.example *An example of an amigaDisk object*

Description

An example of an amigaDisk-class object.

Format

An S4 amigaDisk-class object.

Details

An amigaDisk-class object that represents an 'Old File System' formatted and bootable disk. It is used in multiple examples of this package. It contains a directory structure and some files that can be accessed using this package. The content of this example might change in future versions.

Examples

```
data("adf.example")
```

`adf.file.exists` *Test file or directory existsence in an amigaDisk object*

Description

Tests whether a specific file (or directory) exists in an `amigaDisk` object.

Usage

```
## S4 method for signature 'amigaDisk,character'
adf.file.exists(x, file)
```

Arguments

`x`	An `amigaDisk` object in which this method will check for the file's existence.
`file`	A `character` string representing a file or directory name. Use Amiga specifications for file name (see `current.adf.dir`).

Details

This method will look for a file/directory header, based on its name. If such a header exists, it is assumed that the file exists. The file/directory itself is not checked for validity.

Value

Returns a `logical` value indicating whether the file exists or not.

Author(s)

Pepijn de Vries

Examples

```
data(adf.example)

## This file exists:
adf.file.exists(adf.example, "df0:mods/mod.intro")

## This file doesn't:
adf.file.exists(adf.example, "df0:idontexist")
```

amigaBlock-class *The amigaBlock class*

Description

The Commodore Amiga stores data on floppy disks as 512 byte blocks. This class reflects such a block.

Details

There are several types of blocks. Most important are the boot block (used for booting the Amiga system), the root block (containing information on the disk and the root directory), header blocks (indicating where to find file data) and data blocks (containing the actual file data). See this package's vignette for more details. use the `amigaBlock-method` to extract a specific block from an `amigaDisk` object.

Slots

`data` The `raw` data of a 'block' of data on an Amiga disk file. Each block holds 512 bytes of information. This slot is therefore a `vector` of the same length.

Author(s)

Pepijn de Vries

Examples

```
## create a block with no data:
new("amigaBlock")
```

amigaBlock-method *Extract block from or replace a block on an amigaDisk object*

Description

Extract an `amigaBlock` from an `amigaDisk` object, or replace it on the disk.

Usage

```
## S4 method for signature 'amigaDisk,numeric'
amigaBlock(x, block)

## S4 replacement method for signature 'amigaDisk,numeric,amigaBlock'
amigaBlock(x, block) <- value
```

Arguments

x	An amigaDisk object from which the block needs to be extracted or on which the block needs to be replaced.
block	A numeric identifier (whole numbers ranging from 0 up to 1759 (DD disk) or 3519 (HD disk)).
value	An amigaBlock object with which the block at the specified location on the disk needs to be replaced.

Details

Information is stored in 512 byte blocks on floppy disks. This method extracts a specific block at a numeric identifier (whole numbers ranging from 0 up to 1759 (DD disk) or 3519 (HD disk)) from an amigaDisk object.

Value

The amigaBlock object at the specified location is returned. In case of the replace method, an amigaDisk object with a replaced amigaBlock is returned.

Author(s)

Pepijn de Vries

See Also

Other block.operations: get.blockID, get.diskLocation

Examples

```
## get the root block from the example adf:
amigaBlock(adf.example, 880)

## Create a completely blank disk without file system:
blank.disk <- new("amigaDisk")

## Replace the boot block on the blank disk with
## that from the example object:
amigaBlock(blank.disk, 0) <- amigaBlock(adf.example, 0)

## The blank disk now has a boot block,
## but still no file system...
```

amigaDateToRaw *Convert date time objects into raw values*

Description

This function converts date-time objects into raw data conform Amiga file system specifications.

Usage

```
amigaDateToRaw(x, format = c("long", "short"), tz = "UTC")
```

Arguments

x	A (vector of) POSIXt object(s).
format	a character string indicating whether the date should be stored as short or long integers.
tz	A character string specifying the time zone to be used to convert the date time object. Note that the time zone is not stored on the Amiga. By default the Universal time zone (UTC) is assumed. You will get a warning when you use a timezone other then UTC.

Details

The Amiga file system stores date time objects as three unsigned short (16 bit) or long (32 bit) integers. Where the values are number of days, minutes and ticks (fiftieth of a second) since 1978-01-01 respectively.

As these values are always positive, only date time values on or after 1978-01-01 are allowed. The inverse of this function can be achieved with rawToAmigaDate.

Value

returns raw data reflecting the date-time objects conform the Amiga file system specifications.

Author(s)

Pepijn de Vries

See Also

Other raw.operations: amigaIntToRaw, displayRawData, rawToAmigaDate, rawToAmigaInt, rawToBitmap

Examples

```
## Note that using the same date-time with different timezones will
## result in different raw data. The time zone is not stored.
amigaDateToRaw(as.POSIXct("1978-01-01 13:30", tz = "UTC"))
amigaDateToRaw(as.POSIXct("1978-01-01 13:30", tz = "CET"))
```

amigaDisk-class *The amigaDisk class*

Description

An S4 class representing the information from an Amiga Disk File.

Details

An Amiga Disk File (ADF) holds the raw data of an amiga disk in the same order as blocks (amigaBlock-class) on the physical disks. As an Amiga Disk can hold any kind of information, so can this class.

An ADF file does not hold any other information. The size of the file will dictate whether it represents a double density floppy disk (880 kB) or a high density floppy disk (1760 kB). The disk type is also stored in this class.

Finally, the current directory is stored with this class. Which is only useful for DOS-formatted disks (with a file structure). By default this is set to the disk's root.

For more (technical) backgrounds please check this package's vignette.

Use the objects constructor (`new("amigaDisk")`) to create a completely blank disk (without a filesystem). If you want to be able to transfer files from and to the virtual disk, use `blank.amigaDOSDisk` instead.

Slots

`data` The `raw` data of the virtual disk. It should be a `vector` of length 901,120 in case of a double density disk and 1,802,240 in case of a high density disk.

`type` A `character` indicating whether the virtual disk represents a `"DD"` (double density, most common) or `"HD"` (high density) disk.

`current.dir` An `integer`, pointing at the block address of the current directory of this virtual disk. Use `current.adf.dir` to get or set the current directory.

Author(s)

Pepijn de Vries

Examples

```
## This creates a blank non-bootable, non-DOS disk:
new("amigaDisk")
```

amigaIntToRaw *Convert Amiga integers into raw values*

Description

Convert 8, 16, or 32-bit signed or unsigned integer values into raw data, conform Amiga specifications.

Usage

```
amigaIntToRaw(x, bits = 8, signed = F)
```

Arguments

x	A vector of class numeric which needs to be converted into raw values.
bits	Number of bits that represents the integer value. Should be 8 or a positive multitude of 8.
signed	A logical value indicating whether the numeric values is signed (TRUE, default) or not (FALSE).

Details

The Commodore Amiga has specified the following data formats to represent integer data: BYTE (signed 8-bit integer), UBYTE (unsigned 8-bit integer), WORD (signed 16-bit integer), UWORD (unsigned 16-bit integer), LONG (signed 32-bit integer), ULONG, (unsigned 32-bit integer). This function converts such integers into raw data.

Value

Returns (a vector of) raw data, representing the integer value(s) conform Amiga specifications.

Author(s)

Pepijn de Vries

See Also

Other raw.operations: amigaDateToRaw, displayRawData, rawToAmigaDate, rawToAmigaInt, rawToBitmap

Examples

```
## some unsigned 8-bit integers:
ubyte <- sample.int(255, 100, TRUE)

## The same values as raw data:
amigaIntToRaw(ubyte)

## some signed 8-bit integers:
```

```
byte <- sample.int(255, 100, TRUE) - 128

## The same values as raw data:
amigaIntToRaw(byte, signed = TRUE)

## some signed 16-bit integers:
word <- sample.int(2^16, 100, TRUE) - 2^15

## The same values as raw data:
amigaIntToRaw(word, 16, TRUE)

## note that 16-bit integers require
## twice as many raw values:
length(amigaIntToRaw(word, 16, TRUE))
length(amigaIntToRaw(byte, 8, TRUE))
```

`blank.amigaDOSDisk` *Create blank disk with file system*

Description

Create a virtual blank DOS formatted floppy disk with a file system on it.

Usage

```
## S4 method for signature 'character'
blank.amigaDOSDisk(diskname, disktype = c("DD", "HD"),
  filesystem = c("OFS", "FFS"), international = F, dir.cache = F,
  bootable = T, creation.date = Sys.time())
```

Arguments

`diskname`	A `character` string of the desired disk name. Disk name should be between 1 and 30 characters long, and should not contain any colon or forward slash characters.
`disktype`	Either "`DD`" (double density, most common and therefore default) or "`HD`" (high denisity). The type of disk the blank disk should represent.
`filesystem`	Either "`OFS`" (old file system) or "`FFS`" (fast file system). `FFS` is not compatible with Amiga OS <2.0. On the original system, the FFS was slightly faster and can requires less data for the file system. It is however less robust: on corrupt disks, file recovery is more difficult.
`international`	The international mode was introduced in Amiga OS 2.0. In lower versions, international characters were mistakenly not converted to uppercase when comparing file names. The international mode (set this argument to `TRUE`) corrects this mistake. The international mode is not compatible with Amiga OS <2.0.

`dir.cache` The directory cache mode (set this argument to `TRUE`) was introduced with Amiga OS 3.0 (and is not compatible with lower versions). On real machines this allowed for slightly faster directory listing (but costs disk space). The directory cache mode is always used in combination with the 'international mode'.

`bootable` When this argument is set to `TRUE`. Minimal executable code is added to the bootblock. This code will open the command line interface when the disk is used to boot the system. In Amiga OS >2.0, the 'Startup-Sequence' file needs to be present for this, otherwise the screen will remain black on booting. See also the `boot.block.code` data.

`creation.date` A `POSIXt` object. Will be used and stored as the creation date of the virtual disk. Note that the Amiga does not store the time zone and UTC is assumed as default. The Amiga stores the date and time as positive integers, relative to 1st of January in 1978. As a result, dates before that are not allowed.

Details

Creates a blank `amigaDisk` object. This method differs from the object constructor (`new("amigaDisk")`) because it also installs a file system on the disk. The blank disk can thus be used to write files onto, and is also usable in Amiga emulators. For use in emulators, the object needs to be saved with the `write.adf` method.

Value

Returns a blank `amigaDisk` object with a file system installed on it.

Author(s)

Pepijn de Vries

Examples

```
## Create a blank virtual disk compatible with
## Amiga OS 1.x and up (Note that spaces in file and
## disk names are allowed but not recommended):
disk.os1x <- blank.amigaDOSDisk(diskname = "I'm_OS_1.x_compatible",
                                disktype = "DD",
                                filesystem = "OFS",
                                international = FALSE,
                                dir.cache = FALSE,
                                bootable = TRUE)

## create a disk that is compatable with OS 2.x and up
## (no backward compatability):
disk.os2x <- blank.amigaDOSDisk(diskname = "I'm_OS_2.x_compatible",
                                disktype = "DD",
                                filesystem = "FFS",
                                international = TRUE,
                                dir.cache = FALSE,
                                bootable = TRUE)
```

```
## create a disk that is compatable with OS 3.x and up
## (no backward compatability):
disk.os3x <- blank.amigaDOSDisk(diskname = "I'm_OS_3.x_compatible",
                                disktype = "DD",
                                filesystem = "FFS",
                                international = TRUE,
                                dir.cache = TRUE,
                                bootable = TRUE)
```

`boot.block.code` *Minimal executable code for a bootblock*

Description

A minimal piece of code required to boot to a command line interface on a Commodore Amiga.

Format

A data frame with two columns. The first column contains the assembled code (as `raw` data). The second column contains the corresponding Motorola 68000 (the main CPU of the original Commodore Amiga) assembly syntax.

Details

The first two blocks (`amigaDisk`) are special and are called the boot block. This block should contain information on the type of disk and possibly some executable code that will be run at boot time. This `data.frame` contains some minimal executable code that will start the Amiga command line interface. On Amiga OS >=2.0 the screen will stay black unless a startup-sequence file is present on the disk.

The original code is from Thomas Kessler as published by Laurent Clévy (http://lclevy.free.fr/adflib/adf_info.html).

Examples

```
data("boot.block.code")

## To create a basic boot block for a DD disk:
bblock <- new("amigaBlock", data =
  c(as.raw(c(0x44, 0x4F, 0x53, 0x00, 0xE3, 0x3D, 0x0E, 0x73,
  0x00, 0x00, 0x03, 0x70)), unlist(boot.block.code$assembled),
  raw(419))
)
## The raw data preceding the executable code are
## a label, flags and a checksum.
```

current.adf.dir *Get or set the current directory of an amigaDisk object*

Description

Get or set the current directory of an amigaDisk object.

Usage

```
## S4 method for signature 'amigaDisk'
current.adf.dir(x)

## S4 replacement method for signature 'amigaDisk,character'
current.adf.dir(x) <- value
```

Arguments

x	An amigaDisk object for which the current directory needs to be obtained or changed.
value	A character representation of the path, that needs to be set as current directory. Use Amiga DOS syntax as specified in the details

Details

By default the disk's root is stored as the current directory for a new amigaDisk object. With this method, the current directory can be retrieved or changed.

For this purpose the path should be specified conform Amiga DOS syntax. Use the disk's name or "DF0" followed by a colon in order to refer to the disk's root. Subdirectories are separated by forward slashes ("/"). Colons and forward slashes are not allowed in file and directory names. Both upper and lowercase letters are allowed in file and directory names. The case is ignored when identifying files however. This packages will NOT follow the Amiga's full search path (http://wiki.amigaos.net/wiki/AmigaOS_Manual:_AmigaDOS_Working_With_AmigaDOS#Search_Path).

Value

Returns a character representation of the current directory.

Author(s)

Pepijn de Vries

Examples

```
data(adf.example)

## by default the current dir is the
## disk's root. The disk name is
```

```
## therefore shown when running
## current.adf.dir for the provided
## example data:

current.adf.dir(adf.example)

## change the current dir:
current.adf.dir(adf.example) <- "DF0:this/is/a/deep/path"

## confirm that it has changed:
current.adf.dir(adf.example)

## let's set it back to the disk's root:
current.adf.dir(adf.example) <- "DF0:"
```

`dir.create.adf` *Create a directory on an amigaDisk object*

Description

Create a directory on a virtual amiga floppy disk represented by an `amigaDisk` object.

Usage

```
## S4 method for signature 'amigaDisk,character,missing,missing'
dir.create.adf(x, path, date,
  comment)

## S4 method for signature 'amigaDisk,character,POSIXt,missing'
dir.create.adf(x, path, date,
  comment)

## S4 method for signature 'amigaDisk,character,POSIXt,character'
dir.create.adf(x, path, date,
  comment)
```

Arguments

`x`	An `amigaDisk` on which the directory should be created.
`path`	Specify the directory that should be created on `x`. You can specify the full path on the virtual disk conform Amiga DOS syntax (see `current.adf.dir` details). When no full path is specified the new directory will be created in the current directory. Note that wild cards are not allowed.
`date`	A `POSIXt` object that will be used as the directory modification date. When missing the system time will used.
`comment`	An optional `character` string that will be included in the directory header as a comment. Should not be longer than 79 characters.

Details

Create a directory on a virtual amiga floppy disk represented by an amigaDisk object. Make sure that the virtual disk is DOS formatted.

Value

Returns an amigaDisk object on which the directory is created.

Author(s)

Pepijn de Vries

Examples

```
## Not run:
## create a blank DOS disk:
blank.disk <- blank.amigaDOSDisk("blank", "DD", "FFS", TRUE, FALSE, FALSE)

## creating a new directory on the blank disk is easy:
blank.disk <- dir.create.adf(blank.disk, "new_dir")

## in the line above, the directory is placed in the
## current directory (the root in this case). Directories
## can also be created by specifying the full path:

blank.disk <- dir.create.adf(blank.disk, "DF0:new_dir/sub_dir")

## check whether we succeeded:
list.adf.files(blank.disk)

## we can even make it the current dir:
current.adf.dir(blank.disk) <- "DF0:new_dir/sub_dir"

## End(Not run)
```

---

displayRawData *Display raw data in a comprehensive way*

Description

Cat raw data to the sink in columns with ASCII code

Usage

```
displayRawData(x, ncol = 4, col.wid = 4, address.len = 3, hex.upper = T)
```

Arguments

x	A vector of class raw to be displayed.
ncol	Number of columns of hexadecimal code to display.
col.wid	Width of each column (in bytes) to display.
address.len	Length of the hexadecimal address (in number of hexadecimal digits) to display.
hex.upper	logical value, to specify whether hexadecimals should be displayed in uppercase (TRUE, default) or lowercase (FALSE).

Details

As binary data is hard to decipher this function will cat raw data as hexadecimal code in columns, together with the relative (hexadecimal) address of the data and an ASCII translation of the data. Hexadecimals are shown in space separated columns for improved readability. Special characters are replaced by dots in the ASCII representation.

Raw data is padded with zeros at the end to fill remaining columns...

Value

The character string send to the sink is also returned by the function.

Author(s)

Pepijn de Vries

See Also

Other raw.operations: amigaDateToRaw, amigaIntToRaw, rawToAmigaDate, rawToAmigaInt, rawToBitmap

Examples

```
## Display some raw random data:
displayRawData(as.raw(sample.int(100)))

## Display the full ASCII table:
displayRawData(as.raw(0:255))
```

---

get.adf.file *Get a file from an amigaDisk object*

Description

Get files stored on virtual amigaDisks as raw data or copy as file.

Usage

```
## S4 method for signature 'amigaDisk,character,missing'
get.adf.file(x, source, destination)

## S4 method for signature 'amigaDisk,character,character'
get.adf.file(x, source, destination)

## S4 method for signature 'amigaDisk,character,ANY'
get.adf.file(x, source, destination)
```

Arguments

x	An amigaDisk object from which a file needs to be extracted.
source	Specify the source file's path on the amigaDisk object, conform Amiga specs. See current.adf.dir for details on these specs.
destination	either a file name or a file connection, that allows writing binary data (see e.g., file or url). When the destination is missing, the file content is returned as raw data.

Details

Amiga DOS formatted disks can store any kind of file (as long as the disk's capacity allows it). Use this method to extract such files embedded in an Amiga Disk File (ADF) as raw data or copy to a file on your system.

Value

Returns a vector of raw data when the argument destination is missing. Otherwise returns nothing.

Author(s)

Pepijn de Vries

Examples

```
data(adf.example)

## Not run:
## get the file "Startup-Sequence" from the virtual
## example disk and save as a text file in the
## current working directory:
get.adf.file(adf.example, "DF0:S/Startup-Sequence", "startup.txt")

## End(Not run)

## get the same file as raw data
## by omitting the destination:
startup <- get.adf.file(adf.example, "DF0:S/Startup-Sequence")
```

```
## Look, it's a text file:
cat(rawToChar(startup))

if (requireNamespace("ProTrackR", quietly = TRUE)) {
  ## look there is a typical ProTracker module on
  ## the example disk. You can load it like this:

  ## get the file from the virtual disk
  ## as raw data
  mod.raw <- get.adf.file(adf.example, "DF0:mods/mod.intro")

  ## open a raw connection with the
  ## newly imported raw data
  con <- rawConnection(mod.raw, "rb")

  ## and read it as a ProTracker module
  mod <- ProTrackR::read.module(con)
  close(con)

  ## plot the first sample from the module:
  plot(ProTrackR::waveform(ProTrackR::PTSample(mod, 1)),
       type = "l", ylab = "amplitude")
} else {
  cat("You need to install and load the\nProTrackR package for this part of the example.")
}
```

`get.blockID` *Get the block ID from the physical location on the disk*

Description

Get the block identifier based on the physical location on a disk (side, cylinder and sector) and the disk type.

Usage

```
## S4 method for signature 'character,numeric,numeric,numeric'
get.blockID(disktype, sector,
  side, cylinder)
```

Arguments

`disktype`	A `character` string indicating the type of disk: `DD` for double density disks. `HD` for high density disks.
`sector`	`numeric` identifier for the sector on the disk, ranging from 0 up to 10 (`DD` disks) or 21 (`HD` disks).
`side`	`numeric` identifier for the side of the disk (0 or 1).
`cylinder`	`numeric` identifier for the cylinder on the disk, ranging from 0 up to 79.

Details

Data on Amiga floppy disks are stored as 512 byte blocks. These blocks are physically stored on a specific cylinder and side at a specific sector. This method returns the block identifier based on the physical location on the disk. The inverse of this function is achieved with the `get.diskLocation` method.

Note that all identifiers (or indices) have a base at zero, for consitency with Amiga specifications and documentation, opposed to the base of one used in R.

Value

Returns the `numeric` identifier for the corresponding block.

Author(s)

Pepijn de Vries

See Also

Other block.operations: `amigaBlock-method`, `get.diskLocation`

Examples

```
## Get the block identifier for sectors 0 up to 3 combined with
## cylinders 0 up to 3 on side 0 of the disk:
get.blockID(disktype = "DD",
            sector   = 0:3,
            side     = 0,
            cylinder = 0:3)
```

`get.diskLocation` *Get the physical location on the disk for a specific block*

Description

Get the side, cylinder and sector on a disk, based on disk type and block id.

Usage

```
## S4 method for signature 'character,numeric'
get.diskLocation(disktype, block)
```

Arguments

`disktype`	A `character` string indicating the type of disk: `DD` for double density disks. `HD` for high density disks.
`block`	`numeric` identifier of a block. Whole numbers ranging from 0 up to 1759 (for DD disks) or 3519 (for HD disks). Note that the base index is zero (for consistency with Amiga specifications and documentation) opposed to the base of one used in R.

Details

Data on Amiga floppy disks are stored as 512 byte blocks. These blocks are physically stored on a specific cylinder and side at a specific sector. This method returns the identifiers for the physical location based on the block identifier. The inverse of this function is achieved with the `get.blockID` method.

Value

Returns a `list` with corresponding sector, side and cylinder identifiers (`numeric`).

Author(s)

Pepijn de Vries

See Also

Other block.operations: `amigaBlock-method`, `get.blockID`

Examples

```
## get the physical location of the first 20 blocks on a DD disk
## and arrange as a data.frame:
as.data.frame(get.diskLocation("DD", 0:19))
```

`is.amigaDOS` *Check if amigaDisk object is DOS formatted*

Description

This method checks if there is a DOS file structure is present on the `amigaDisk` object.

Usage

```
## S4 method for signature 'amigaDisk'
is.amigaDOS(x)
```

Arguments

`x`	An `amigaDisk` object for which the check should be performed.

Details

Not all Amiga Disk Files have a DOS file structure on them. This function checks if there is.

Value

Returns a `logical` value, indicating whether the disk is DOS formatted. When it is not, the attributes to the returned value will contain information as to why the disk is not DOS compatible.

Author(s)

Pepijn de Vries

Examples

```
data(adf.example)

## let's check if the example amigaDisk object
## is DOS formatted:

is.amigaDOS(adf.example)

## it apparently is
```

---

is.bootable *Check if amigaDisk object is bootable*

Description

This function checks if the amigaDisk object represents a bootable disk.

Usage

```
## S4 method for signature 'amigaDisk'
is.bootable(x)
```

Arguments

x	An amigaDisk object for which the check should be performed.

Details

The first two amigaBlock-class objects on a disk are special and are called the boot block. The boot block will determine whether an Amiga can boot from the disk.

This function will determine whether the Amiga would attempt to execute the machine code present on the boot block. It will not check whether it would be successful at that, as that would require emulation of the Commodore Amiga system.

Value

Returns a logical value, indicating whether the disk is bootable.

Author(s)

Pepijn de Vries

Examples

```
data(adf.example)

## let's check if the example amigaDisk object
## is bootable:

is.bootable(adf.example)

## it apparently is
```

`list.adf.files` *List files in an amigaDisk directory*

Description

Get a list of files in a specific directory on a virtual `amigaDisk`.

Usage

```
## S4 method for signature 'amigaDisk,missing'
list.adf.files(x, path)

## S4 method for signature 'amigaDisk,character'
list.adf.files(x, path)
```

Arguments

`x`	An `amigaDisk` object for which the files should be listed.
`path`	Specify the path on the `amigaDisk` object, conform Amiga specs, for which files should be listed. See `current.adf.dir` for details on these specs.

Details

As an analogue of `list.files`, this method list files in a specific directory. But in this case the files are located on a virtual floppy disk represented by the `amigaDisk` object. This works only for DOS-formatted (`is.amigaDOS`) virtual disks.

Value

Returns a `vector` of `characters` listing the files in the specified directory on the virtual disk.

Author(s)

Pepijn de Vries

Examples

```
data(adf.example)

## show all files in the root of the example
## disk file:
list.adf.files(adf.example)

## you can also list the files in a specified
## directory:
list.adf.files(adf.example, "DF0:mods")

## For the same path, only now specified
## relatively to the current directory:
list.adf.files(adf.example, "mods")
```

`print` *Print Amiga Disk File objects*

Description

A method to print Amiga Disk File S4 class objects to the sink.

Usage

```
## S4 method for signature 'amigaDisk'
print(x, ...)

## S4 method for signature 'amigaBlock'
print(x, ...)
```

Arguments

`x`	Either a `amigaDisk` or `amigaBlock` object.
`...`	further arguments passed to or from other methods

Value

Returns nothing (`NULL`).

Author(s)

Pepijn de Vries

Examples

```
data(adf.example)

print(adf.example)
```

put.adf.file *Put a file onto an amigaDisk object*

Description

Put a file onto a virtual amiga floppy disk represented by an amigaDisk object.

Usage

```
## S4 method for signature 'amigaDisk,raw,character,POSIXt,character'
put.adf.file(x, source,
  destination, date, comment)

## S4 method for signature 'amigaDisk,raw,character,POSIXt,missing'
put.adf.file(x, source,
  destination, date, comment)

## S4 method for signature 'amigaDisk,raw,character,missing,missing'
put.adf.file(x, source,
  destination, date, comment)

## S4 method for signature 'amigaDisk,character,character,POSIXt,character'
put.adf.file(x,
  source, destination, date, comment)

## S4 method for signature 'amigaDisk,character,character,POSIXt,missing'
put.adf.file(x, source,
  destination, date, comment)

## S4 method for signature 'amigaDisk,character,character,missing,missing'
put.adf.file(x,
  source, destination, date, comment)

## S4 method for signature 'amigaDisk,character,missing,missing,missing'
put.adf.file(x, source,
  destination, date, comment)
```

Arguments

x	An amigaDisk onto which the file should be put.
source	Either a character string of the source file's path; or a vector of raw data that should be written to the destination file. Wildcards are not allowed (see details)
destination	A character string of the destination path on the virtual floppy disk where the source file should be put. The path should be conform Amiga specs (see

	current.adf.dir). When the destination is missing or only specifies a directory, the file will be put into the current directory (current.adf.dir) or specified path of x respectively. In that case, the same file name as that of the source file is used. Wild cards are not allowed (see details).
date	A POSIXt object that will be used as the file modification date. When missing the system time will used.
comment	An optional character string that will be included in the file header as a comment. Should not be longer than 79 characters.

Details

Put a file or raw data from your local system onto a virtual amiga floppy disk represented by an amigaDisk object. Make sure that the virtual disk is DOS formatted. This method can only put one file at a time onto the virtual virtual disk. It is therefore not allowed to use wild cards in the source or destination names. Use loops to add multiple files onto a virtual disk.

Value

Returns an amigaDisk object onto which the source file is put at the specified destination.

Author(s)

Pepijn de Vries

Examples

```
## Not run:
## create a blank disk to put files onto:
blank.disk <- blank.amigaDOSDisk("blank", "DD", "OFS", TRUE, FALSE, FALSE)

## let's copy the base package 'INDEX' file onto the
## virtual disk:
blank.disk <- put.adf.file(blank.disk, system.file("INDEX"))

## We can also put raw data onto the virtual disk:
blank.disk <- put.adf.file(blank.disk, raw(2048), "DF0:null.dat")

## check whether we succeeded:
list.adf.files(blank.disk)

## End(Not run)
```

---

rawToAmigaDate *Convert raw values into a date time object*

Description

This function converts raw data into a date time object conform the Amiga file system specifications.

Usage

```
rawToAmigaDate(x, format = c("long", "short"), tz = "UTC")
```

Arguments

x	a vector of raw values with a length of a multitude of 6 (for the short format) or 12 (for the long format).
format	a character string indicating whether the date is stored as short or long integers.
tz	A character string specifying the time zone to be used to retrieve the date time object. Note that the time zone is not stored on the Amiga. By default the Universal time zone (UTC) is assumed.

Details

The Amiga file system stores date time objects as three unsigned short (16 bit) or long (32 bit) integers. Where the values are number of days, minutes and ticks (fiftieth of a second) since 1978-01-01 respectively.

As these values are always positive, only date time values on or after 1978-01-01 are allowed. The inverse of this function can be achieved with amigaDateToRaw.

Value

Returns a POSIXct object based on the provided raw data.

Author(s)

Pepijn de Vries

See Also

Other raw.operations: amigaDateToRaw, amigaIntToRaw, displayRawData, rawToAmigaInt, rawToBitmap

Examples

```
## all raw data is zero, so the origin date is returned:
rawToAmigaDate(raw(12))

## let's get the date, one day, one minute and 50 ticks from the origin:
rawToAmigaDate(amigaIntToRaw(c(1, 1, 50), 32))
```

rawToAmigaInt *Convert raw values into Amiga integers*

Description

Convert raw data into 8, 16, or 32-bit signed or unsigned integer values, conform Amiga specifications.

Usage

```
rawToAmigaInt(x, bits = 8, signed = F)
```

Arguments

x	A vector of class raw to be converted into a character.
bits	Number of bits that represents the integer value. Should be 8 or a positive multitude of 8.
signed	A logical value indicating whether the integer should be signed (TRUE, default) or not (FALSE).

Details

The Commodore Amiga has specified the following data formats to represent integer data: BYTE (signed 8-bit integer), UBYTE (unsigned 8-bit integer), WORD (signed 16-bit integer), UWORD (unsigned 16-bit integer), LONG (signed 32-bit integer), ULONG, (unsigned 32-bit integer). This function converts raw data into such integers. Note that WORD and UWORD are also referred to as SHORT and USHORT respectively.

Value

A numeric value (or a vector of values), representing the integer data represented by the provided raw data. Note that R defines integer as 32-bit signed integers and cannot store the 32-bit signed values. Therefore a numeric value is returned rather than an explicit integer.

Author(s)

Pepijn de Vries

See Also

Other raw.operations: amigaDateToRaw, amigaIntToRaw, displayRawData, rawToAmigaDate, rawToBitmap

Examples

```
## Let's start by obtaining unsigned 8-bit integers:
rawToAmigaInt(as.raw(0:255))

## note that this is the same as:
as.numeric(as.raw(0:255))

## but with this function we can also get signed values:
rawToAmigaInt(as.raw(0:255), signed = TRUE)

## Furthermore 32 or 16-bit integers can also be obtained.
## Let's look at 16-bit integers:
rawToAmigaInt(as.raw(0:255), 16)

## Note that 16-bit integers require twice as many bytes
## as 8 bit integers:
length(rawToAmigaInt(as.raw(0:255), 16))
length(rawToAmigaInt(as.raw(0:255), 8))
```

`rawToBitmap` *Convert raw data into a bitmap or vice versa*

Description

Convert raw data into a bitmap or vice versa (i.e., binary data) conform Amiga specifications.

Usage

```
rawToBitmap(x, invert.bytes = F, invert.longs = T)

bitmapToRaw(x, invert.bytes = T, invert.longs = T)
```

Arguments

`x`	A `vector` of `raw` data, in case `rawToBitmap` is used. A `vector` of `raw`, `interger` or `logical` values should be used in case of `bitmapToRaw`. In the latter case each value in the `vector` is interpeted as a bit and should be a mutiple of 8 long.
`invert.bytes`	A `logical` value. When set to `TRUE`, the bit order of bytes are reversed.
`invert.longs`	A `logical` value. When set to `TRUE`, the bit order of long values (32 bits) are reversed. When `x` does not have a multiple length of 32 bits or 4 bytes, `x` will be padded with zeros to the right, but the result will be trimmed to correspond with the length of `x`. Note that data might get lost this way.

Details

A bitmap is simply put a map of bits (binary data, which can be interpeted as 0 or 1; or FALSE and TRUE). Bitmaps can have several purposes, also on the Commodore Amiga. The Amiga file system uses a bitmap to indicates which blocks are occupied with data and which are free. Bitmaps can also be used in bitmap images where each bit indicates which color should be used for a specific pixel in an image. These function can be used to convert raw data into usable bitmaps or vice versa.

As the Commodore Amiga is a big-endian system (most significant bit first) using a 32 bit CPU, it may sometimes necessary to invert the bits of a byte or longs (4 bytes, 32 bits), which can be done with the arguments 'invert.bytes' and 'invert.longs' respectively.

Value

Returns a vector of raw data in case of bitmapToRaw, and a vector of binary raw values in case of rawToBitmap.

Author(s)

Pepijn de Vries

See Also

Other raw.operations: amigaDateToRaw, amigaIntToRaw, displayRawData, rawToAmigaDate, rawToAmigaInt

Examples

```
## The bitmap block of the example disk is located at block
## number 882 (note that this is not true for all disks,
## the actual location is stored in the root block)
data(adf.example)
bitmap.block <- amigaBlock(adf.example, 881)

## bitmap data are stored in bytes 5 up to 224 in this block:
bitmap.raw <- bitmap.block@data[5:224]

## let's get the bitmap from the raw data:
bitmap <- rawToBitmap(bitmap.raw)

## Whe can now get the occupied blocks (minus one is used for
## the discrepancy in indexing):
which(bitmap != as.raw(0x01)) - 1

## we can also do the reverse:
bitmap.raw.new <-  bitmapToRaw(bitmap)
## it should be the same as the original raw data:
all(bitmap.raw.new == bitmap.raw)

## WARNING: don't use these methods to directly
## modify an amigaDisk objects bitmap block. The
## file system on that object may get corrupted.
## All methods in this package should update the
## bitmap block automatically and cleanly...
```

`read.adf` *Read an Amiga Disk File*

Description

Read data from an Amiga Disk File (ADF) to an `amigaDisk` object. Alternatively data can be read from an ADZ file.

Usage

```
## S4 method for signature 'character'
read.adf(file)

## S4 method for signature 'ANY'
read.adf(file)

## S4 method for signature 'character'
read.adz(file)
```

Arguments

`file`	Either a file name or a file connection, that allows reading binary data (see e.g., `file` or `url`). `read.adz` only accepts file names.

Details

Amiga Disk Files usually have a .adf-extension to the file name. It should be 880 kB (double density) or 1760 kB (high density) in size. This function can read such files.

Alternatively, ADZ files can also be read. These are essentially gzipped ADF files.

Note that this package cannot read extended ADF files containing information on the disk's Modified frequency modulation (MFM). This information is typically only required for copy protected disk's and is therefore out of the scope of this package.

Value

Returns an `amigaDisk` object read from the provided amiga disk file

Author(s)

Pepijn de Vries

See Also

Other io.operations: `write.adf`

Examples

```
## Not run:
## In order to read an adf-file, we first need one.
## so let's first write the example obect to a file:
data(adf.example)

## write it to the current working directory:
write.adf(adf.example, "test.adf")

## now we can read it again:
my.disk <- read.adf("test.adf")
print(my.disk)

## and this is how you read it,
## using a connection:
con <- file("test.adf", "rb")
my.disk2 <- read.adf(con)
close(con)

print(my.disk2)

## Alternatively, you can work with ADZ files:
write.adz(adf.example, "test.adz")
my.disk3 <- read.adz("test.adz")

print(my.disk3)

## End(Not run)
```

`write.adf` *Write an amigaDisk object to an ADF file*

Description

Write an `amigaDisk` object to an Amiga Disk File (ADF) or alternatively to an ADZ file.

Usage

```
## S4 method for signature 'amigaDisk,ANY'
write.adf(x, file)

## S4 method for signature 'amigaDisk,character'
write.adf(x, file)

## S4 method for signature 'amigaDisk,character'
write.adz(x, file)
```

Arguments

x	An amigaDisk object that needs to be saved to an ADF file.
file	either a file name to write to, or a file connection, that allows to write binary data (see file). write.adz only accepts a file name.

Details

Use this function to write amigaDisk objects as binary data to so-called Amiga Disk Files (ADF). These files can be used as input for Amiga emulator software.

Alternatively, the object can be saved with 'write.adz', which is essentially a gzipped version of an ADF file.

Value

Writes to an ADF file but returns nothing.

Author(s)

Pepijn de Vries

See Also

Other io.operations: read.adf

Examples

```
## Not run:
## Let's write the example data to an ADF file:
data(adf.example)

## Let's put it in the current working directory:
write.adf(adf.example, "test.adf")

## You can also use file connections to do the same:
con <- file("test2.adf", "wb")
write.adf(adf.example, con)
close(con)

## Last but not least the same object can be saved
## as an adz file:
write.adz(adf.example, "test.3.adz")

## End(Not run)
```

Index