Package 'serial'

September 11, 2017

Type Package

Title The Serial Interface Package

Version 1.3

Date 2017-09-11

Author Martin Seilmayer

Maintainer Martin Seilmayer <m.seilmayer@hzdr.de>

Description Provides functionality for the use of the internal hardware for RS232/RS422/RS485 and any other virtual serial interfaces of the computer.

Depends R (>= 2.15.0)

License GPL-2

RoxygenNote 5.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2017-09-11 08:59:08 UTC

R topics documented:

close.serialConnection

Function to close an serial interface.

Description

This function closes the corresponding connection.

Usage

```
## S3 method for class 'serialConnection'
close(con, ...)
```

Arguments

con	serial connection
...	is ignored

See Also

serialConnection

isOpen

Generic function for isOpen

Description

Generic function for isOpen

Usage

```
isOpen(con, ...)
```

Arguments

con	connection Object
...	not used

isOpen.default *Default function from base-package*

Description

Default function from base-package

Usage

```
## Default S3 method:
isOpen(con, rw = "")
```

Arguments

con	connection object
rw	defines the mode of operation

See Also

isOpen

isOpen.serialConnection *Tests whether the connection is open or not*

Description

Tests whether the connection is open or not

Usage

```
## S3 method for class 'serialConnection'
isOpen(con, ...)
```

Arguments

con	connection of the class serialConnection
...	not used

Value

returns {F, T} for 'not open' and 'is open'

`listPorts` *Lists the serial interfaces.*

Description

This function lists all installed serial interfaces in a computer. Thereby Windows, Linux and MacOs behave a little bit different. Please ensure that you have the appropriate permissions to do a search in the registry or in the corresponding linux folders.

Usage

```
listPorts()
```

Value

A character vector with the list of comports is returned.

Windows

In a Windows environment, this function tries to read out the registry keys located in:

`"HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\SERIALCOMM"`

This should be consistent with all installed hardware ports plus all virtual ports.

Linux and MacOS

Here the situation is a bit different, compared to Windows. All possible serial devices are located in `"/dev/tty[...]"` as a file connection. Still, all virtual and closed dev's can be found here. This is confusing, because one will find more devices in this folder than physically (virtual) present. In addition to that, on Ubuntu linux systems in `"/sys/devices/pnp0/..."` only the plug and play devices of interest are listed again. That is the reason why, the function returns a subset of `"/dev/tty[...]"`, which is also present in the `"../pnp0/.."` folder.

On MacOs the installed interfaces are marked by `"tty.<name>"`, with a unique name after the dot, which makes it easier to search for installed devices.

Subsequently, the user must know which interface is present and which isn't. AND the user must have at least reading permissions in the corresponding folders. So in the end, this function is a best guess of what is installed.

open.serialConnection *Function to initialize an serial interface.*

Description

This function initializes the serial interface and opens it for later usage.

Usage

```
## S3 method for class 'serialConnection'
open(con, ...)
```

Arguments

con	serial connection
...	is ignored

See Also

serialConnection

read.serialConnection *Reads from the serial interface.*

Description

This function reads from the serial interface as long as the buffer is not empty. The read takes place per byte.

Usage

```
read.serialConnection(con)
```

Arguments

con	serial connection

Value

The result is a string, which can be converted to raw as necessary

See Also

serial

Examples

```
# See the top package documentation
```

serial *A serial communication interface for R.*

Description

This R package provides the functionality to use the serial communication ports "COM" or "tty" to use the RS232/RS422/RS485 functionality of the corresponding hardware. Also virtual COM-ports via USB do work, as long as they are mapped to COM[n] (win) or tty[n] (Linux) in the operating system.

`open(con)` opens a serial connection

`close(con)` closes the serial connection

`read.serialConnection(con)` byte wise read from the interface as long as the buffer is empty

`write.serialConnection(con,dat)` writes a string to the serial interface

`isOpen(con)` test a connection, whether it is open or not

`listPorts()` list all available ports on the system

Examples

```
# for this example I used the 'null-modem' emulator 'com0com' for Windows
# which is available on 'http://com0com.sourceforge.net/'
# Here the pair of com-ports is 'CNCA0' <-> 'CNCB0'

# Test the functionality:
# ======================
#
# first: install the virtual null-modem connection like
#        com0com (win) or tty0tty (linux)
#        Hint: Some unix insist on port names like 'ttyS[n]'.
#
# second: setup a terminal program (like HTerm or gtkterm) and listen to
#         com-port 'CNCB0' (or what ever you have installed)
#         or (for unix only) 'cat /dev/tnt1' will output tnt1 to console

## Not run:

# Now configure one of the com-ports with appropriate connection properties
con <- serialConnection(name = "testcon",port = "CNCA0"
                        ,mode = "115200,n,8,1"
                        ,newline = 1
                        ,translation = "crlf"
                        )

# let's open the serial interface

open(con)

# write some stuff
```

```
write.serialConnection(con,"Hello World!")

# read, in case something came in
read.serialConnection(con)

# close the connection
close(con)

## End(Not run)
```

`serialConnection` *Sets up the interface parameters.*

Description

This is the constructor of the serial interface connection.

Usage

```
serialConnection(name, port = "com1", mode = "115200,n,8,1",
  buffering = "none", newline = 0, eof = "", translation = "lf",
  handshake = "none", buffersize = 4096)
```

Arguments

`name`	optional name for the connection
`port`	comport name; also virtual com's are supported; maybe USB schould work too
`mode`	communication mode '<BAUD>, <PARITY>, <DATABITS>, <STOPBITS>'
	`BAUD` sets the baud rate (bits per second)
	`PARITY` *n, o, e, m, s* stands for 'none', 'odd', 'even', 'mark' and 'space'
	`DATABITS` integer number of data bits. The value can range from 5 to 8
	`STOPBITS` integer number of stop bits. This can be '1' or '2'
`buffering`	'none', for RS232 serial interface, other modes don't work in this case
`newline`	<BOOL>, whether a new transmission starts with a newline or not.
	`TRUE` **or 1** send newline-char according to `<translation>` befor transmitting
	`FALSE` **or 0** no newline
`eof`	<CHAR>, termination char of the datastream. It only makes sense if `<translation>` is 'binary' and the stream is a file
`translation`	each transmitted string is terminated by the transmission character. This could be 'lf', 'cr', 'crlf', 'binary'
`handshake`	determines the type of handshaking the communication
	'`none`**'** no handshake is done
	'`rtscts`**'** hardware handshake is enabled
	'`xonxoff`**'** software handshake via extra characters is enabled
`buffersize`	defines the system buffersize. The default value is 4096 bytes (4kB).

Details

Linux and Windows behave a little bit different, when utilizing serial com ports. Still, by providing the name (like 'COM1' or 'ttyS1') and the appropriate settings, the serial interface can be used. Even virtual com ports, like the FTDI usb uart chips will work, as long they map to a standard serial interface in the system.

Since the `serial` package relies on R's built in Tcl/Tk engine the configuration of the serial port takes place in the Tcl framework. This becomes important when different buffer sizes are set. For Windows the Tcl "-sysbuffer" parameter is invoked, whereas on unix-like systems "-buffersize" does the job.

Value

An object of the class '`serialConnection`' is returned

`write.serialConnection` *Wirtes data to serial interface.*

Description

Wirtes data to serial interface.

Usage

```
write.serialConnection(con,dat)
```

Arguments

`con`	serial Connection
`dat`	data string to write on the serial interface. At the moment this must be a string `'...'`. See examle section in `serial`.

See Also

`serial`

Examples

```
# See the top package documentation

## Not run: write.serialConnection(con, 'Hello World!')
```

Index