Package 'serial'

May 9, 2026

Type Package

Title The Serial Interface Package

Version 3.0

Author Martin Seilmayer

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

Description Enables reading and writing binary and ASCII data to RS232/RS422/RS485 or any other virtual serial interface of the computer.

Depends R (>= 2.15.0)

License GPL-2

RoxygenNote 7.1.0

NeedsCompilation no

Repository CRAN

Date/Publication 2020-05-14 20:00:02 UTC

Contents

close.serialConnection 2
flush.serialConnection 2
isOpen 3
isOpen.default 3
isOpen.serialConnection 4
listPorts 4
nBytesInQueue 5
open.serialConnection 6
print.serialConnection 6
read.serialConnection 7
serial 8
serialConnection 9
summary.serialConnection 11
write.serialConnection 12

Index **13**

`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`

`flush.serialConnection` *Flushes the connection.*

Description

Some times (and depending on buffering mode) the connection buffer needs to be flushed manually. This command empties the buffer by sending all remaining bytes.

Usage

```
## S3 method for class 'serialConnection'
flush(con)
```

Arguments

`con`	serial connection

Value

Nothing is returned

See Also

`serial`

Examples

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

`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 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 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, Ubuntu linux systems do list the plug and play devices of interest in "/sys/devices/pnp0/..." 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>" or "cu.<name>", with a unique name after the dot, which makes it easier to search for installed devices. Here a tty device is a modem which waits for a DCD (Data Carrier Detect) signal to receive data. The device is blocked as long such a signal is not detected. The corresponding cu device manages the out going communication for historical reasons. However a cu port is not blocked and should be used in favour on MacOS.

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.

---

nBytesInQueue *Reads out the number of characters / bytes pending in the input and output buffer*

Description

Reads out the number of characters / bytes pending in the input and output buffer

Usage

```
nBytesInQueue(con)
```

Arguments

con	serial connection

Value

named vector with number of bytes

See Also

serial

Examples

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

`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`

`print.serialConnection`
Print method for serialConnection

Description

Outputs major information of the serial connection.

Usage

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

Arguments

`x`	`serialConnection` Object
`...`	not used

Value

nothing to return

`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 line in normal operation mode. Here end-of-line characters (\n;\r) are clipped according to the settings (`translation`).

Usage

```
read.serialConnection(con,n = 0)
```

Arguments

`con`	serial connection
`n`	number of bytes to read. Only in binary mode. `n=0` (default) reads the whole buffer at once.

Details

In binary (hex-) mode the read takes place per byte. The result is a raw vector of hexadecimal numbers. To get numeric values `as.numeric` function must be invoked. Mind: Values form 0x01 – 0x31 might be displayed as escaped characters like "\001" if they are interpreted as string. If the end-of-file character specified by `eof` is received the reading stops. A `close(con)` – `open(con)` sequence must be invoked to reopen the connection. If `n>0` <n> bytes will be read. In case of less than `n` bytes available the function returns the buffer without waiting for all `n` characters. If the result is empty (zero length) then the empty string ” is returned in ASCII mode or `NA` in binary mode.

Value

In normal mode the result is a string. In binary mode raw values will be returned

See Also

`serial`

Examples

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

`serial` *A serial communication interface for R.*

Description

This R package provides the functionality of serial communication ports "COM" or "tty" to use the RS232/RS422/RS485 capabilities 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

`flush(con)` flushes the serial connection

`nBytesInQueue(con)` get the length of pending input an output queue

`read.serialConnection(con)` read from the interface as long as the buffer is not empty

`write.serialConnection(con,dat)` writes a data (character or binary) 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)

# show summary
summary(con)

# close the connection
close(con)



# Reading and writing binary (hexadecimal) data
# remember: Everything is a string, so you might need data conversation

con <- serialConnection(name = "testcon",port = "CNCA0"
                        ,mode = "115200,n,8,1"
                        ,translation = "binary" # switches to binary data
                        )

# let's open the serial interface

open(con)

# write some stuff
write.serialConnection(con, rawToChar(as.raw(15)) ) # 0x0F
write.serialConnection(con, c(15,20) ) # 0x0F, 0x14
write.serialConnection(con, c(0x6F,0x6C) )

# read, in case something came in
# the output is always a character vector
a <- read.serialConnection(con)

# convert the character vector to hexadecimal (raw) values
print(a)

# 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 = "auto",
  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* corresponds to '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', best for RS232 serial interface. Connection buffer is flushed (send) when ever a write operation takes place. 'line', buffer is send after newline character (\n or 0x0A) is recognized. 'full' write operations will be bufferd until a flush(con) is invoked.
newline	<BOOL>, whether a transmission ends 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 (end-of-file). It only makes sense if <translation> is 'binary' and the stream is a file. Must be in the range of 0x01 – 0x7f. When the conection is closed eof is send as the last and final character.
translation	Determines the end-of-line (eol) character and mode of operation. This could be 'lf', 'cr', 'crlf', 'binary', 'auto' (default). A transmission is complete if eol symbol is received in non binary mode.
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', 'ttyS1' or 'cu.<name>') 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

Binary Data

Handling binary data is possible by setting `transaltion = 'binary'`. Pay attention that input and output vectors are characters with a number range of 0...0xFF which might require certain conversations e. g. `charToRaw()` or `rawToChar()` functions. If `eof`-character is defined, this symbol terminates the input data stream. Every byte in the buffer after that symbol is deleted/ignored. The next transmission is valid again up to that symbol. If the connection is closed `eof` is send to terminate the output data stream. Remind, the `newline` option works here too. It adds a line feed or `0x0A` - byte to the end of each output respectively.

ASCII Data

In non binary mode, ASCII-communication is assumed. This means, that each string, which is send or received, carries valid 8bit ASCII characters (0x01 – 0xFF). Some of these characters appear as escaped sequences, if they are not printable. A string is terminated by the end-of-line character (e. g. \n). The transmission ends and so becomes valid if the symbol is detected according to the `translation` setting. Sending terminated strings invokes the substitution of the end-of-line character according to the `translation` setting.

`summary.serialConnection` *Serial Connection Summary*

Description

Displays summarized informations and status of the serial connection.

Usage

```
## S3 method for class 'serialConnection'
summary(object, ...)
```

Arguments

object	object of type serialConnection
...	not used

Value

Table of connection properties

---

write.serialConnection

Wirtes data to serial interface.

Description

Writes to a serial connection in ascii or binary mode.

Usage

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

Arguments

con	serial Connection
dat	data string to write on the serial interface. This must be a string '...' in case of ascii communication. In case of binary communication also numeric vectors are allowed. See examle section in serial.

Details

In normal operation mode (non-binary ascii mode) write.serialConnection respects the translation and adds the end-of-line characters (\n;\r) according to the settings. Any input is converted to character, i.e. c(1,2,3) becomes '123' and so on.

In binary mode *no* end-of-line characters are added. The input argument must be of type raw or string. If dat is numeric (vector) it is converted to raw.

Value

The status of success 'DONE' or 'Nothing to do' is returned.

See Also

serial

Examples

```
# See the top package documentation

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

Index

∗ **RS232**
serial, 8
∗ **RS422**
serial, 8
∗ **RS485**
serial, 8
∗ **USB**
serial, 8
∗ **serial communication**
serial, 8

close.serialConnection, 2

flush.serialConnection, 2

isOpen, *3*, 3
isOpen.default, 3
isOpen.serialConnection, 4

listPorts, 4

nBytesInQueue, 5

open.serialConnection, 6

print.serialConnection, 6

read.serialConnection, 7

serial, *2*, *5*, *7*, 8, *12*
serialConnection, *2*, *6*, 9
summary.serialConnection, 11

write.serialConnection, 12