DevicePlugin

SerialCom

Revision 2022.05.20

Table of Contents

1 Description

Universal serial communication via standard hardware/virtual serial ports (COM).

Features:
- Read/write data (text, binary)
- Read buffer (text, binary)
- Hardware signals control

2 Commands

2.1 *idn? (Identification)

```
*idn?
```

Gets the plug-in identification string.

Parameters

No parameters.

Return value

The identification string in standard format "`<company>,<product/name>,<serial-no>,<version>`".

2.2 set (Set global parameters)

```
set:{RecvMode=[enum]}{;Async=[bool]}{;WrEnding=[string]}{;RdEnding=[string]}
      {;RdTimeout=[int]}{;WrTimeout=[int]}{;AutoDiscard=[bool]}
```

Set global parameters: asynchronous mode, line endings and timeouts.

Parameters

`RecvMode` *(deprecated, replaced by "Async")*	[enum]	Receiving mode: • `discard` - All incoming data are discarded • `buf`, `bytebuf`/`linebuf` - Buffering incoming data • `request` - No data are buffered, communication style is request/answer
`Async`	[bool]	• `true` = asynchronous mode (buffering, read/readln not allowed) • `false` = synchronous mode (no buffering, read/readln possible)
`WrEnding`	[string]	Line-ending character(s) to be used by writeln command ("`\n`" by default)
`RdEnding`	[string]	Line-ending character(s) to be used by readln and request commands ("`\n`" by default)
`AutoDiscard` (or `AutoDiscardInBuffer`)	[bool]	Clear byte and line buffers at the start of any writing operating (write, writeln and request)
`RdTimeout`	[int]	Receive timeout, 10 to 25000 [ms]. Applied to read, readln and request commands. (2000ms by default)
`WrTimeout`	[int]	Write timeout, 10 to 25000 [ms]. Applied to write, writeln and request commands. (2000ms by default)

`Rts`	[bool]	Manual control of RTS line. Possible when no handshake is set (RTS/CTS).
`Dtr`	[bool]	Manual control of DTR line.

Note: if any parameter is not passed, the value is not changed, not replaced by default.

Return value

No return value

Examples

`set:Async=true:WrEnding="\r":RdEnding="\r":RdTimeout=3000`
Set asynchronous mode, write and read endings to "\r" (CR) character and read-timeout to 3 sec.

`set:rts=true:dtr=true`
Set both RTS and DTR line active (ON).

2.3 open (Connect to port)

```
open:{port=[string]}{;baudrate=[int]}{;databits=[int]}{;stopbits=[enum]}
      {;parity=[enum]}{;handshake=[enum]}
```

Connect to default pre-configured port or to specified port using specified paramters.

Parameters

`port`	[string]	Target port to connect, i.e. "COM1". **Default:** (from pre-configuration)
`baudrate`	[int]	Baudrate [bps], 300 to 115200. Typical baudrates are 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600 and 115200 bps. **Default:** (from pre-configuration)
`databits`	[int]	Number of data bits, 5 to 8. Typical frame consists of 8 data-bits. **Default:** (from pre-configuration)
`stopbits`	[enum]	Number of stop bits: • `1` • `1p5` (or `1.5`) • `2` Typical frame has 1 stop bit. **Default:** (from pre-configuration)
`parity`	[enum]	Parity, the additional computed frame's bit: • `none` • `odd` • `even` **Default:** (from pre-configuration)
`handshake`	[enum]	Flow control: • `none` - no flow control • `xonxoff` - software flow control using reserved characters Xon and Xoff • `rtscts` - hardware flow control using RTS/CTS singnals **Default:** (from pre-configuration)

Return value

No return value

Examples

```
open
```
Connect to port using all default pre-configured parameters.

```
open: port=COM1
```
Connect to specified port "COM1", all other parameters are taken from pre-configuration.

```
open: port=COM2; baudrate=19200; stopbits=1p5; parity=even; handshake=xonxoff
```
Connect to COM2 at 19200 bps, with 1.5 stop bits, even parity and software Xon/Xoff handshake. The number of data-bits is taken from device configuration.

2.4 close (Disconnect from port)

```
close
```

Disconnect from currently opened COM port.

Parameters

No paramters

Return value

No return value

2.5 write/writeln (Write/write-line)

```
write: <str>{: ashex=[ bool]}
writeln: <str>
```

Send string via opened COM port. The `writeln` command appends the `WrEnding` string automatically.

Parameters

`str`	[string]	String to be sent. Use escape sequences to send non-printable characters.
`asHex`	[bool]	All 2-characters will be converted treated as hex string. Length of the str parameter must be even. **Default:** false

Return value

No return value

Examples

```
write: "abc 123"
```
Send 7 bytes: "`abc 123`". Quotes are recommended because of space character.

```
write: \xAA
```
Send 1 byte: 0xAA (10101010 in binary). Escape-sequence is required, because the 0xAA is non-printable character.

```
write: "\x1Babc\x00"
```
Send 5 bytes: `0x1B` (ESC) + "`abc`" + `0x00` (NULL). Quotes are recommended because of more characters and escape-sequences (but not required).

`writeln:abc123`
Send 7 bytes: "`abc123\n`". The "`\n`" is the currently set `WrEnding` string.

`write:"002255FF71"`
Send 5 bytes: "0x00 0x22 0x55 0xFF 0x71".

2.6 read/readln (Read/read-line)

```
read{:<count>}{:timeout=[int]}{:ashex=[bool]}
readln{:timeout=[int]}
```

Read data from serial-port buffer:
- all (`read` without count parameters)
- specified count (`read` with count parameters)
- until `RdEnding` string has been received (`readln`)

Cannot be used when asynchronous mode is active. See "async" paramter of set command.

Parameters

`count`	[int]	Number of bytes to be read for serial-port buffer. The reading is blocked until specified number of bytes has been read (timeout limited), otherwise the timeout error is thrown. If the parameter is not present, all available data are read at the time of reading (it is possible to get the empty string - no data). **Default:** (all data)
`timeout`	[string]	Maximum time to wait until specified number of bytes are received (`read`) or read ending-string is received (`readln`). **Default:** timeout set by the `RdTimeout` parameter of set command
`ashex`	[bool]	All bytes/characters will be converted into hex string. **Default:** false

Return value

Received data. Non-printable characters are replaced by escape-sequences.

Examples

`read`
Read all available data.

`read:count=20`
Waits until 20 bytes are received for the default timeout

`readln`
Waits until read ending-string is received and return received data without the ending-string. Timeout is the default.

`readln:timeout=5000`
Same like the "readln" above, but with overriding the timeout to 5 seconds.

`read:count=4:ashex=true`
Waits until 20 bytes are received for the default timeout. If the received data was 'idn?', the returned result will be ' 69646E3F'.

2.7 request (Request/answer)

```
request:<str>{:wrline=[bool]}{:read=[int]}{:repeat=[int]}{:interval=[int]}{:accept=[string
:<str>{:...}
```

Synchronous request/answer communication: write string to COM and waits for a reply for a defined timeout. Starting the command by a colon ": " only is the short way (alias) to call `request` command - see examples below.

Asynchronous mode is temporarily disabled until request is done (or finished by timeout). Then the previous state is restored.

Parameters

`str`	[string]	String to be sent. Use escape sequences to send non-printable characters.
`wrline`	[bool]	If true, append the write ending-string (WrEnding). **Default:** true
`read`	[int]	If the parameter is: • present - wait for a specified number of bytes, range 1 to N • non-present (default) - wait for the read ending-string (RdEnding)
`repeat`	[int]	Number of repeats after receive timeout or failed accept filter. When repeat occurs, param 'str' will be write again, before waiting for received data. **Default:** 0
`interval`	[int]	Interval between repetition. When repeat is zero, this parameter is ignored. **Default:** 0
`accept`	[string]	Accept filter for received data - to compare is used Regex. **Default:** nothing

Return value

Reply to request, without the ending string. Non-printable characters are replaced by escape-sequences.

Examples

Note: examples below expects the "`\n`" to be set for both `WrEnding` (write ending-string) and `RdEnding` (read ending-string).

```
request:*idn?
```
Send 6 bytes: "`*idn?`" + "`\n`" ending-string and waits for answer with the read ending string

```
:*idn?
```
The same like "`request:*idn?`" command above, but using the short way of `request` command.

```
request:"abc":wrline=false
```
Send 3 bytes: "`abc`", without the ending string.

```
request:"cmd\x00":wrline=false:read=10
```
Send 4 bytes: "cmd" + NULL character (without the ending string) and waits for 10 bytes to be received (fixed length).

```
request:"*IDN?":repeat=2:interval=100:accept="^\\w+\\.\\d{3}\\.\\d{2}"
```
Send "*IDN?" to the line and read the response - after successful reading it compares with regex accept filter.
If data match regex, function returns received data.

If data not match regex, whole operation send-read-compare will be repeated maximally two times (timeout or not match error can arise after this).
If data not read because of timeout, command exit immediately with error.

2.8 linebuf (Line-buffer control)

Requires asynchronous mode. See "async" paramter of set command.

```
linebuf:clear
```

Clears all lines from the buffer.

Parameters

No parameters

Return value

No return value

```
linebuf:count
```

Get number of lines in the buffer.

Parameters

No parameters

Return value

Number of lines.

```
linebuf:get=[var]
linebuf:pop=[var]
```

Get or pop line(s) from the buffer.

Parameters

`get`	[var]	Get lines: • `all` - get all lines in the buffer • `rem` - the remainder (currently receiving string before the new-line string is received) • number(s), separated by comma "," - get selected lines from the end of the buffer, range 0 to (count - 1), where 0 is the last received line
`pop`	[var]	Pop (get and remove) lines: • `all` - get all lines in the buffer • `first` - get first read (time) line in the buffer • number(s), separated by comma "," - get selected lines from the end of the buffer, range 0 to (count - 1), where 0 is the last received line

Return value

Line(s) from the string buffer. More than one lines are separated by new-line character "\n" (ASCII 10 = LF).

Examples

For example, lets have following line in the buffer:
- `abc`
- `def`
- `ghi`
- `jkl` (last received)

`linebuf:get=all`
Returns "`abc\ndef\nghi\njkl`" (4 strings, separated by "\n" character).

`linebuf:get=0,2`
Returns "`jkl\ndef`" (2 strings, separated by "\n" character).

`linebuf:get=0`
Returns last received string "`jkl`".

`linebuf:savetofile=[string]`

Save content of line buffer to a text-file.

Parameters

`savetofile`	[string]	Target text filename. The file is overwritten if exists.

Return value

No return value

Examples

`linebuf:savetofile="c:\\lines.txt"`
Saves all lines to "c:\lines.txt" file.

2.9 bytebuf (Byte-buffer control)

Requires asynchronous mode. See "async" paramter of set command.

`bytebuf:clear`

Clears all bytes from the buffer.

Parameters

No parameters

Return value

No return value

`bytebuf:count`

Get number of bytes in the buffer.

Parameters

No parameters

Return value

Number of lines.

`bytebuf: remove=[ array]`

Remove specified byte ranges from the buffer.

Parameters

`remove`	[array] (comma-separated numbers)	Range(s) to remove in format "$\texttt{index}_0$, $\texttt{count}_0$, $\texttt{index}_1$, $\texttt{count}_1$, ..., $\texttt{index}_N$, $\texttt{count}_N$". Where: • `index` = zero-based start index of byte in the buffer, range 0 to (count - 1) • `count` = number of bytes to remove from specified index, range 1 to (count) If there is only one index, it is possible to leave count undefined => 1 byte is removed.

Return value

No return value

Examples

For example, lets have following 16 bytes in the buffer (hex-notation)
`00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff`

`bytebuf: remove=1,3,6,8`
Remove 3 bytes at position 1 and 8 bytes at position 6: `00` ~~`11 22 33`~~ `44 55` ~~`66 77 88 99 aa bb cc dd`~~ `ee ff`

`bytebuf: remove=12`
Remove 1 byte at position 14: `00 11 22 33 44 55 66 77 88 99 aa bb` ~~`cc`~~ `dd ee ff`

`bytebuf: get=[ var]`

Get line(s) from the buffer.

Parameters

`get`	[var]	Get lines: • `all` - get all bytes in the buffer • number(s), separated by comma "," - get selected bytes from the start of the buffer, range 0 to (count - 1)

Return value

Hex-encoded bytes with fixed length of 2 per byte, without any separation (1 byte = 2 letters, 2 bytes = 4 letters, 5 bytes = 10 letters).

Examples

For example, lets have following 16 bytes in the buffer (hex-notation)
`00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff`

`linebuf: get=all`

Returns "`00112233445566778899aabbccddeeff`" (16 bytes = 32 letters).

`linebuf:get=2,3,6,7`
Returns "`22336677`" (4 bytes = 8 letters).

```
bytebuf:savetofile=[string]
```

Save content of byte buffer to a binary-file.

Parameters

`savetofile`	[string]	Target binary filename. The file is overwritten if exists.

Return value

No return value

Examples

`bytebuf:savetofile="c:\\bytes.bin"`
Saves all bytes to "c:\bytes.bin" file.

2.10 trig (String trigger)

```
trig:str=[string]
```

Set a flag when a specified string is received.

Parameters

`str`	[string]	String to set the flag, to deactivate trigger set this string to an empty string ("").

Return value

No return value

Examples

`trig:str="enter:"`
Set the string which cause trig to "`enter:`" (quotes required because of double-colon ":" character).

`trig:str=""`
Disable the previously set string trigger.

```
trig
trig?
```

Get the trigger state.

Parameters

No parameters

Return value

Returns the trigger state as "`0`" (the string has not been received yet) or "`1`" (received).