Teunis van Beelen
Electronics engineer

RS-232 for Linux and Windows

Here you can find code to use the serial port.
It has been tested with GCC on Linux and MinGW on Windows.
Handshaking or flowcontrol is not supported.
It uses polling to receive characters from the serial port.
Interrupt/event-based is not supported.
Without modifications, this code uses 8N1 (8 databits, no parity, 1 stopbit).

No serial port available on your computer? Use a USB to RS232 cable.

Download

Januari 31, 2014 new version:

- Fixed a bug that made it impossible to read from the serial port on Linux 64-bit systems.

December 26, 2013 new version:

- added the function RS232_IsDCDEnabled()

February 1, 2013 new version:

- added the prefix "RS232_" to all functions in order to prevent clashes with other libraries
- set the DTR pin and RTS pin active when opening a serial port (some RS-422/485 converters need this to enable the outputbuffers)
- added the baudrates 500000 and 1000000 for windows, this can be usefull when using an FTDI-chip or USB-converter
- added the devices "/dev/ttyAMA0" and "/dev/ttyAMA1" for use with the Raspberry Pi
- added the devices "/dev/ttyACM0" and "/dev/ttyACM1" for use with the Atmel (USB-)microcontrollers
- added the devices "/dev/rfcomm0" and "/dev/rfcomm1" for use with Bluetooth
- added the devices "/dev/ircomm0" and "/dev/ircomm1" for Infrared communication
- added the following functions: RS232_enableDTR(), RS232_disableDTR(), RS232_enableRTS(), RS232_disableRTS() and RS232_IsDSREnabled()

- changed function "cprintf()" to "RS232_cputs()"

The sourcecode

- RS-232.tar.gz

Extract the file and copy rs232.h and rs232.c into your project directory.
Include rs232.h in your program sourcecode (like: #include "rs232.h") and compile and link rs232.c (add rs232.c to your project).

Functions

int RS232_OpenComport(*int comport_number*, *int baudrate*)

Opens the comport, comportnumber starts with 0 (see the list of numbers).
Baudrate is expressed in baud per second i.e 115200 (see the list of possible baudrates).
Returns 1 in case of an error.

int RS232_PollComport(*int comport_number*, *unsigned char *buf*, *int size*)

Gets characters from the serial port (if any). Buf is a pointer to a buffer
and size the size of the buffer in bytes.
Returns the amount of received characters into the buffer.
After succesfully opening the COM-port, connect this function to a timer.
The timer should have an interval of approx. 100 milliSeconds.
Do not forget to stop the timer before closing the COM-port.

int RS232_SendByte(*int comport_number*, *unsigned char byte*)
Sends a byte via the serial port. Returns 1 in case of an error.

int RS232_SendBuf(*int comport_number*, *unsigned char *buf*, *int size*)

Sends multiple bytes via the serial port. Buf is a pointer to a buffer
and size the size of the buffer in bytes.
Returns -1 in case of an error, otherwise it returns the amount of bytes sent.
This function blocks (it returns after all the bytes have been processed).

void RS232_CloseComport(*int comport_number*)

Closes the serial port.

void RS232_cputs(*int comport_number*, *const char *text*)

Sends a string via the serial port. String must be null-terminated.

The following functions are normally not needed but can be used to set or check the status of the control-lines:

void RS232_enableDTR(*int comport_number*)

Sets the DTR line high (active state).

void RS232_disableDTR(*int comport_number*)

Sets the DTR line low (non active state).

void RS232_enableRTS(*int comport_number*)

Sets the RTS line high (active state).

void RS232_enableRTS(*int comport_number*)

Sets the RTS line low (non active stae).

int RS232_IsDSREnabled(*int comport_number*)

Checks the status of the DSR-pin. Returns 1 when the the DSR line is high (active state), otherwise 0.

int RS232_IsCTSEnabled(*int comport_number*)

Checks the status of the CTS-pin. Returns 1 when the the CTS line is high (active state), otherwise 0.

int RS232_IsDCDEnabled(*int comport_number*)

Checks the status of the DCD-pin. Returns 1 when the the DCD line is high (active state), otherwise 0.

Notes:

You don't need to call RS232_PollComport() when you only want to send characters.
Sending and receiving do not influence eachother.

The os (kernel) has an internal buffer of 4096 bytes.
If this buffer is full and a new character arrives on the serial port,
the oldest character in the buffer will be overwritten and thus will be lost.

After a successfull call to RS232_OpenComport(), the os will start to buffer incoming characters.

Example code that demonstrates how to use the library to receive characters and print them to

the screen:

(compile with the command: gcc main.c rs232.c -Wall -o2 -o test)

```
/**************************************************

file: main.c
purpose: simple demo that receives characters from
the serial port and print them on the screen,
exit the program by pressing Ctrl-C

**************************************************/

#include <stdlib.h>
#include <stdio.h>

#ifdef _WIN32
#include <Windows.h>
#else
#include <unistd.h>
#endif

#include "rs232.h"



int main()
{
  int i, n,
      cport_nr=0,        /* /dev/ttyS0 (COM1 on windows) */
      bdrate=9600;       /* 9600 baud */

  unsigned char buf[4096];


  if(RS232_OpenComport(cport_nr, bdrate))
  {
    printf("Can not open comport\n");

    return(0);
  }

  while(1)
  {
    n = RS232_PollComport(cport_nr, buf, 4095);

    if(n > 0)
    {
      buf[n] = 0;   /* always put a "null" at the end of a string! */

      for(i=0; i < n; i++)
      {
        if(buf[i] < 32)  /* replace unreadable control-codes by dots */
        {
          buf[i] = '.';
        }
      }

      printf("received %i bytes: %s\n", n, (char *)buf);
    }

#ifdef _WIN32
    Sleep(100);
#else
    usleep(100000);  /* sleep for 100 milliSeconds */
#endif
  }
```

```
    return(0);
}
```

tip: To get access to the serial port on Linux, you need to be a member of the group "dialout".

Look here for a timer library.

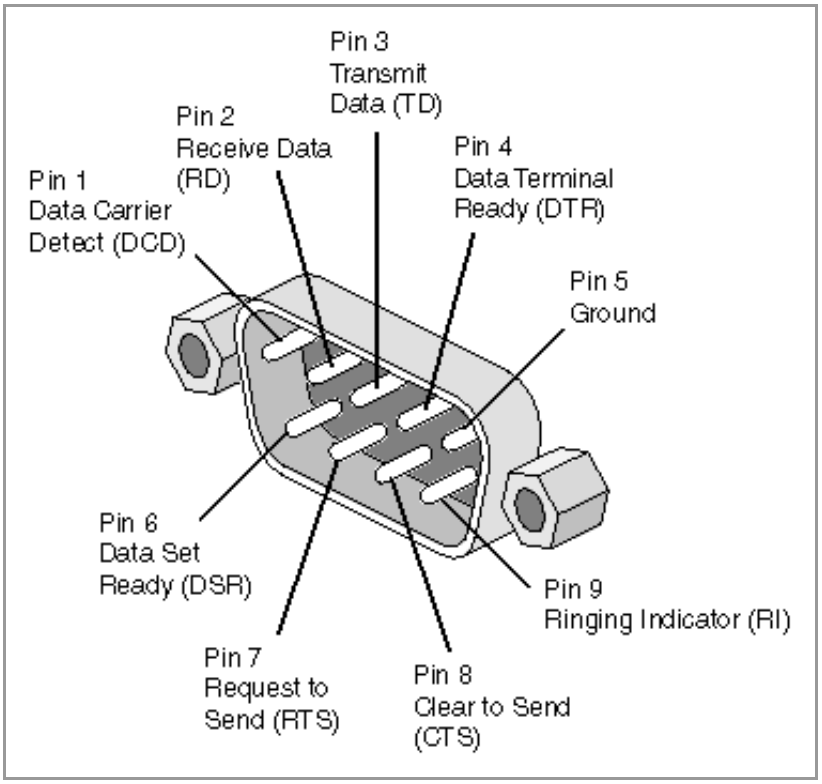
List of comport numbers and possible baudrates:

	Linux	windows
0	ttyS0	COM1
1	ttyS1	COM2
2	ttyS2	COM3
3	ttyS3	COM4
4	ttyS4	COM5
5	ttyS5	COM6
6	ttyS6	COM7
7	ttyS7	COM8
8	ttyS8	COM9
9	ttyS9	COM10
10	ttyS10	COM11
11	ttyS11	COM12
12	ttyS12	COM13
13	ttyS13	COM14
14	ttyS14	COM15
15	ttyS15	COM16
16	ttyUSB0	n.a.
17	ttyUSB1	n.a.
18	ttyUSB2	n.a.
19	ttyUSB3	n.a.
20	ttyUSB4	n.a.
21	ttyUSB5	n.a.
22	ttyAMA0	n.a.

23	ttyAMA1	n.a.
24	ttyACM0	n.a.
25	ttyACM1	n.a.
26	rfcomm0	n.a.
27	rfcomm1	n.a.
28	ircomm0	n.a.
29	ircomm1	n.a.

Linux	windows
50	n.a.
75	n.a.
110	110
134	n.a.
150	n.a.
200	n.a.
300	300
600	600
1200	1200
1800	n.a.
2400	2400
4800	4800
9600	9600
19200	19200
38400	38400
57600	57600
115200	115200
230400	128000
460800	256000
500000	500000
576000	n.a.
921600	n.a.
1000000	1000000

Connector pinlayout



When using this code, you only need to connect pins 2, 3 and 5 of the serial port (plus the shielding) to your device.

home >RS-232 for Linux and WIN32 | feel free to contact me