Serial connection

(Version 1.0 (english language) Auguts 2021)

J.Beushausen@gmx.de

1 Motivation

Again and again questions arise about the interaction between micro-controllers and PC programs. Mostly it is about a serial connection between both devices. The challenge in the realization lies in the mastery of the interaction between the three components hardware, controller programming and PC programming. In the following text such a connection shall be presented exemplarily.

2 Basic

For the PC program Delphi is used, the controller program is created with BASCOM for a processor of the AT-Mega family. Thus two popular development environments are in use. The protocol for transmission is kept as simple as possible. For illustration, first both the client (PC program) and the server (later on the controller in BASCOM) are developed in Delphi on the PC. Of course, the server could be programmed more elegantly on a PC with the available resources and tools, but it shall only serve as an example here and already takes into account the limited possibilities of the controller and the BASIC dialect.

3 Establish connection

Essential for the selection of a physical COM port is its port number. Under Windows the notation is COM1 ... COMxx. The device manager provides information about the interfaces available in the PC. Apart from the "real" COM ports integrated on the motherboard (unfortunately hardly any today) (mostly COM1 and possibly COM2), these can also be USB to serial converters.

Before any program is used at all, a physical connection must first be established between the two devices.

Controllers mostly work with 5V (possibly even with smaller voltages, e.g. 3.3 V). Your built-in serial interface therefore also only supplies voltages of approx. 0 V to approx. 5V. But a (standard) serial connection assumes voltages between approx. -10V and +10 V. For adaptation there are special interface ICs, so called level converters. It is assumed here that the devices to be connected (i.e. PC and PC or PC and controller) both operate with the required levels of approx. +/- 10 V. The serial interface and its specifications are described in detail at https://de.wikipedia.org/wiki/RS-232.

In this example, a connection with only three wires is used.

TX indicates a transmitting line, RX indicates a receiving line and GND stands for the common ground line. A serial interface usually has further connections. They are mainly used to control the data flow. Since this type of control is practically not used by microcontrollers, these connections are not considered further.

The physical connection must now be established as follows.

```
    Device 1                Device 2
(3) TX1 -------------- (2) RX2
(2) RX1 -------------- (3) TX2
(5) GND1 ------------- (5) GND2
```

The values in brackets are the pin numbers on a 9-pin SUB-D connector In addition to the above voltage levels, various other settings must match in order to ensure correct communication.

Baud rate 9600 Baud, Data bits8, Parity none, Stop bit1, flow control none

These are the settings that must be made, for example, for a terminal program (Hyperterminal, Tera Term).

An overview of the details of the various interfaces can be found at http://www.pci-card.com/schnittstellen.html.

4 Transmission protocol

Protocol is the definition of certain properties in the exchange of data. There are a large number of serial protocols.

4.1 Advantages of an ASCII protocol

During the development it is of great advantage to be able to fall back on proven programs (terminal program). Therefore an ASCII protocol is to be used here. With the use of an ASCII protocol errors can be found with it by "looking on it".

4.2 Protocol definition

Practice often requires the parameterization of a process running on a microcontroller. This should be clearly distinguished from a data transfer from or to the controller. In this view, parameters are individual values that fulfill specific tasks, e.g. measuring rate, limit values or conversion factors.

The instructions to the controller are called commands here. The controller must therefore be informed of two things in a command, firstly which parameter it is and secondly the actual value. Furthermore it makes sense to make these parameters readable. The protocol should allow commands with and without parameters.

The following approach has proven to be a simple way to transfer parameters:

send to controller	: capital letter;value CR LF
read from controller	: lower case letter CR LF

The abbreviation CR LF stands for the control characters carriage return and linefeed (ASCII characters 13 and 10).

For smaller tasks the letters can be chosen sensibly, e.g. T for temperature; U for voltage and I for current. It is clear that this kind of naming will lead to conflicts at some point, but on the other hand, important parameter names can be kept in mind, which in turn facilitates the work with a terminal program.

In addition to the letters, the ASCII character set contains sufficient other characters that can be assigned special functions without parameters. For example "?" for querying the firmware version and "!" for a program reset.

The strings "ok" and "error" are used for feedback. A response is only given for those transmissions that do not generate a response themselves.

5 Simulation of the microcontroller program on the PC

The serial interface is accessed via the operating system with the API call CreateFile with various parameters. Since a lot of detail work is required here, other people have thankfully taken the trouble to develop corresponding components. For Delphi there are several components for the communication over the COM port. Exemplarily two sources are mentioned here.

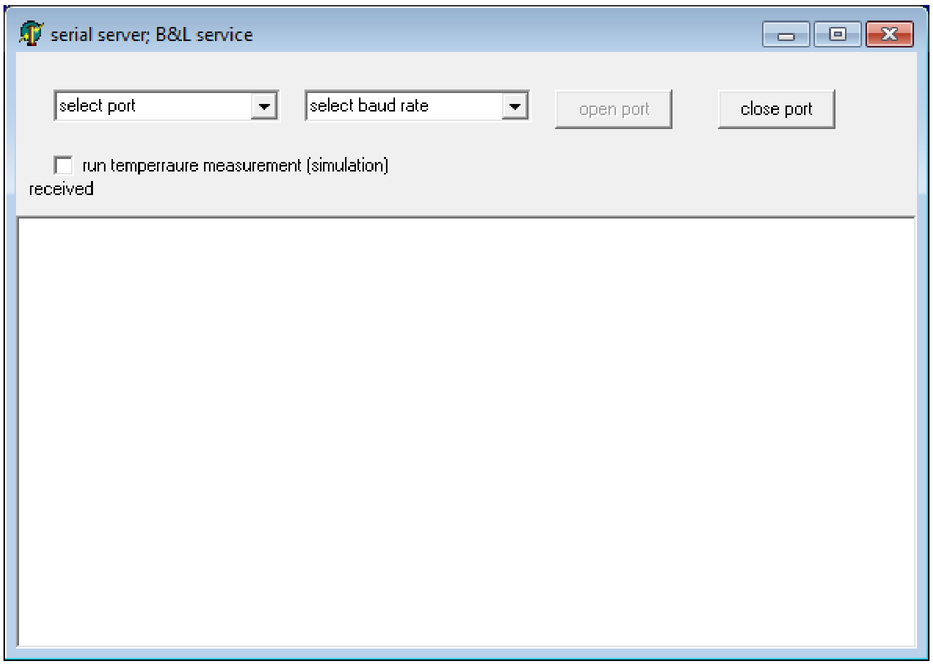
AsyncPro can be downloaded from http://sourceforge.net/projects/tpapro/. This is a very powerful collection of various components.

TComPort is much slimmer. It is available for download at http://sourceforge.net/projects/comport/.

Both components must be installed before they can be used. They facilitate the handling of the COM port extraordinarily and provide very comfortable possibilities.

5.1 Server simulation using

Without actually having a microcontroller available, the functioning of the connection and the protocol can also be followed with two Delphi programs. The server program simulates the behavior of the microcontroller. The user interface of the program is kept very simple, in order not to complicate the view of the essentials by an elaborate design.


serial server; B&L service
select port
select baud rate
open port
close port
run temperraure measurement (simulation)
received

After starting the program the COM port and the baud rate must be selected in two combo boxes. Afterwards the button [open port] is released. A click on [open port] opens the connection. Central element is a field with 26 elements for float numbers (corresponding to A .. Z), in which transmitted values (parameters) are stored.

In the lower area the received characters are displayed.

The program responds to the following commands:

- Capital letter Semicolon Float number CR LF (example X;1234,56 CR LF)
  Reaction : Storage of the float number in a field element under an index (A = 1, B = 2 ...)
- Lower case CR LF (Example x CR LF)
  Reaction : Send float number from a field element. Index (a = 1, b = 2 ...)
- CR LF exclamation mark (example ! CR LF)
  Perform reset, set all field elements to zero or default.

- Dollar sign CR LF (Example $ CR LF)
  Store all field elements permanently. In the microcontroller this will be done later in the EEPROM, the PC program uses an INI file for simulation.
- Question mark CR LF (Example ? CR LF)
  Send version of firmware

5.2 Structure and program flow

The server program is contained in a single unit. In a larger development, for example, all functions that are directly related to the serial interface would be combined in a data module. The component **TComPort is** used.

In the two OnChange events of the combo boxes the port number and the baud rate are set. The actual set functions (*SetComPortNr* and *SetBaudRate*) encapsulate the access to the routines of the TComPort component. This means that the actual program logic does not need to be changed when trying to use alternative components. The incoming characters of the serial interface trigger the OnRxChar event of the component. Here the incoming characters are collected in the string strRX and examined for the character LF. If the character LF is found, the string is examined in *ParseRXStr* for its components.

5.3 Parser

After the incoming string has been received completely, it must be parsed and processed. The function *ParseRXStr is* used for this purpose. Lower and upper case letters, the permitted special characters and the semicolon as separator must be recognized. If a semicolon is recognized, the following part is to be interpreted as a float number. It makes sense to check this substring for permissible rows.

Here, too, only procedures are used that can also be implemented on the microcontroller, e.g. the operator "in" is omitted and instead the individual characters are checked in a loop with the function *isValidValueChar*. Immediately obvious is the admissibility of the digits 0 ... 9 and possibly the signs + and -. But what about an input in scientific notation, e.g. 1.234E-3. So also the character E and depending on the notation also e is allowed. Furthermore, a decimal separator must be considered, which again, depending on the sending system, can be either a point or a comma. In addition still another note: The microcontroller, and/or BASCOM does not stand + at the beginning of the number and expects compellingly a point as decimal separator.

The function *ParseRXStr* now performs the desired decomposition and returns the separated command, the index for the storage and the value. Success or an error is indicated by a corresponding function result.

In case of an error-free decomposition, the received commands are reacted to in a *case* statement and the corresponding actions are triggered. For commands that do not generate their own response, "ok" is sent; in the event of an error, "error" is transmitted.

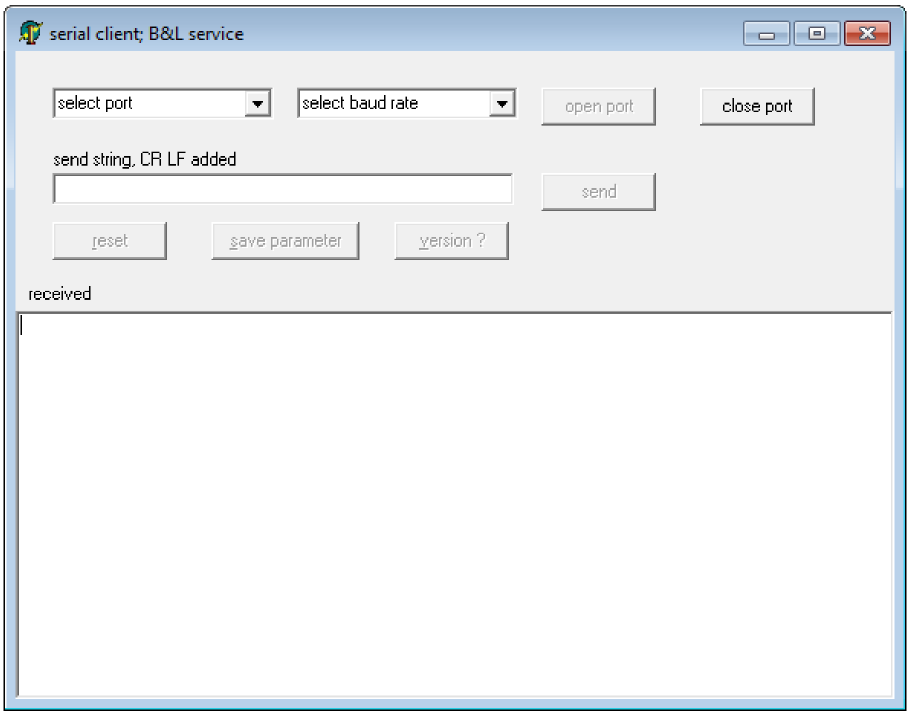
5.4 Simulated temperature measurement

The temperature measurement is arranged in the Application.OnIdle event and is started via a checkbox. An AD converter with 8 bits is simulated and converted into a real temperature using the parameters offset and slope. When the com port is open, this value is then sent as a string. The time interval is set by the procedure *delay*. Also during a simulated measurement the program reacts to commands.

6 Client program

The client program essentially provides the functions of a terminal, specifically tailored to the application at hand.

6.1 Client program using



After starting the program, the COM port and the baud rate must be selected in two combo boxes. Here it should be mentioned again that client and server must be set to the same baud rate but different port numbers. Afterwards the button [open port] is enabled. A click on [open port] opens the connection. Additionally three buttons [reset], [save parameter] and [version ?] are available. They are assigned with the special commands described above, i.e. the characters !, $ and ? Of course, these commands can also be typed into the input field. The required characters CR and LF are added automatically when the [send] button is clicked.

6.2 Structure and program flow

The structure of the client program is much clearer than the structure of the server simulation, since no parser is required here. Incoming characters on the serial interface trigger the *OnRxChar* event of the component. Here the incoming characters are collected in the string strRX and examined for the character LF. If the character LF is found, the string is displayed in the memo field.

7 Microcontroller program

The microcontroller program is written in BASIC and is compiled with the BASCOM compiler. The way in which the program is transferred to the controller will not be discussed here, since this process depends heavily on the available tools (programmer). A demo version of the BASCOM IDE can be downloaded from:

http://www.mcselec.com/index.php?option=com_docman&task=cat_view&gid=99&Itemid=54.

7.1 Initialization

At the very beginning of the program, some settings must be made that relate to the underlying hardware.

The following lines set the processor type (here ATMega32), the used crystal frequency and the baud rate. Furthermore the interrupt routine for the serial interface is assigned and the interrupts are enabled. The setting of the processor type and the crystal frequency must be adapted to the existing hardware.

```
$regfile = "m32def.dat"      ' used processor

$crystal = 14745600          ' used crystal frequency for processor

baud = 9600                  ' baudrate

on urxc Com1_Int             ' assign IRQ service

enable urxc

enable interrupts
```

The definition of various constants and variables follows. Before the start of the main loop, a version identifier is output on the serial interface. This has the advantage that even without functioning communication an output for a first test of the serial connection is always generated.

7.2 Main loop

As usual in microcontroller programs, the processing is done in a large skin loop that starts with the *do* statement and ends with the *loop* statement. Main parts of the loop are the function *ParseRXStr* and the following *case* construct, which evaluates the result of the parser.

7.3 Interrupt handling

Each character arriving at the serial interface triggers an interrupt which is evaluated in the routine *Com1_Int*. Here the arrived character is first buffered in the variable *b_udr1* and then appended to the string *strRX*. If the character LF is recognized as the end character, the variable *b_RxFlag is* set to 1 for notification in the main loop. With this the interrupt processing is finished and kept very short. The actual evaluation then takes place at the head of the main loop. Here strRX is deleted and *b_RxFlag is set to* 0 again.

7.4 Parser

The decomposition and evaluation of the received string is done by the function *ParseRXStr. The* tasks correspond to the description from the section "Simulation on the PC" The two functions *isValidValueChar* and *isValidIndexChar* each examine a character for its permissibility. In the process, *isValidValueChar* also ensures that spaces and plus signs are suppressed. Furthermore, a comma is converted into a period, since this is the correct decimal separator for converting a string into a number.

The following list shows the reaction of the parser when receiving different strings, where the first two receive strings are an empty string and a string with a space respectively, which of course are not visible here in the text.

```
invalid                       index 0 value 0.0                  command 0
invalid                       index 0 value 0.0                  command 0
invalid  X                    index 24 value 0.0                 command 1
invalid  x;                   index 24 value 0.0                 command 2
invalid  X;                   index 24 value 0.0                 command 1
valid  x                      index 24 value 0.0                 command 2
valid  X;1234.56              index 24 value 1234.559936522      command 1
valid  X;-1234.56             index 24 value -1234.559936522     command 1
valid  X;0.1234E-3            index 24 value 0.00012338          command 1
valid  X;+0.1234E-3           index 24 value 0.00012338          command 1
valid  X;+0.1234E+3           index 24 value 123.400001524       command 1
valid  X;-0.1234E+3           index 24 value -123.400001524      command 1
valid  X ;+ 0.12 34E + 3      index 24 value 123.400001524       command 1
```

7.5 Storage

The two functions *LoadParameterFromEEPROM* and *SaveParameterToEEPROM load* or save the parameters from / to the EEPROM (electrically erasable programmable read-only memory) of the processor. This means that the changed parameters are available even after switching off and on. With *ResetParameter* the basic state can be restored on command. In the present example, the parameters S (slope) and O (offset) are then sensibly preassigned.

7.6 Special features during processing

In the interrupt handling a flag is set to signal to the main loop the provision of a received string. The evaluation then takes place during the next run. But what happens if the main loop contains parts with a long processing time? Then the reaction of the controller is delayed accordingly. Since BASCOM does not provide multitasking, this situation cannot be avoided at first. The function *WasteTime()* stands here for a longer processing and waits 1 second. During this time the receive flag is permanently monitored. If a new string has been received, the waiting time stops immediately and the main loop can work. In addition, the function result shows the termination that has occurred. Real processings can also be built up according to this method. Via their function result they signal whether their processing was interrupted or whether the results are usable.

7.7 Temperature measurement

A (simulated) temperature measurement is used here for demonstration. To make the example independent of real hardware, an AD converter with 8 bit resolution is simulated. The raw data have a value range from 0 to 255 and should represent a temperature from -20°C to 120°C. A conversion to real temperatures shall already be done in the controller. For this, under the condition that the conversion has a linear characteristic, two values are needed. These are the slope and the offset of the characteristic curve. Under ideal conditions, both values could be stored as fixed values. In reality, however, transducers and encoders always have tolerances. To take these tolerances into account, the two parameters are to be made loadable. The basic setting is:

Slope	= (120 - (-20)) / 255	= 0.549 °C/bit
Offset		= -20 °C
Temperature		= AD bits * Slope + Offset

The function *GetTempRaw* generates (simulated) values between 0 and 255, ascending with an increment of 4 bits and then descending with a decrement of 2 bits. A global variable (*b_TempSimulation*) is used for this purpose. The function *CalculateTemp* calculates the temperature in degrees Celsius using the stored offset and slope values. The simulated temperature measurement is only executed if the parameter T (i.e. the field element with the index *parTempSimulation*) is greater than zero. In case of a reset, the corresponding field element is set to zero. So from the client must be sent for the start of the measurement e.g. T;1 CR LF.

8 Practical tip

If nothing works at all with a serial interface, proceed as follows:

- use the device manager to check if a COM port is displayed
  (for USB serial converters it may take a long time to install a driver)
- remember a (the) COM port
- Start terminal program with this COM port, switch off flow control
- Connect RX and TX line (on a 9 pin SUB-D this is pin 2 and pin 3)
- Enter characters on the terminal. These characters are then sent via TX and immediately received again at the RX port. If there are no hardware defects, the characters sent must appear (as an echo) in the terminal.

It happens that a USB driver gets "stuck" in such a way that only unplugging and plugging in the converter leads to success.

Since Hyperterminal is no longer included in newer Windows versions, Tera Term can be obtained from here as an alternative. https://en.osdn.jp/projects/ttssh2/releases/

Attention: In the default setting Tera Term only sends CR after an input, if CR LF is to be sent, this must be set separately.

9 Problems when using USB to serial converters or Blue-Tooth

Real serial interfaces (COM port) are rarely found in PCs and notebooks. USB to serial converters are often used as a replacement. They emulate a COM port for the operating system via corresponding drivers. A COM port is also displayed in the device manager. Unfortunately there are a lot of converters with quite different quality of hardware and drivers. To make matters worse, successful use in connection with a particular PC under a particular operating system does not necessarily guarantee successful use on another PC. These findings are not theory, but result from many years of experience of users and own deployments.

Basically you should always realize that ultimately the driver is responsible for the transport of the serial data stream via the USB interface. Thereby the serial data is converted into a USB data stream which is then transferred. If the data rates of the emulated serial interface are not too high, the USB transfer is always "fast enough", but you cannot necessarily rely on exact timing.

Similar considerations apply to the SPP profile (serial port profile) at Blue-Tooth.

https://de.wikipedia.org/wiki/Bluetooth-Profile