# LinTronic

# RS232
# - simplified

## and why USB sucks

| Date | Revised |
|---|---|
| yymmdd | Text |
| 120618 | First release |
| 160306 | Page 4 inserted |

**The purpose of this document is to enable even non-technicals to get 95% of all RS232 installations to work.**

**We will assist you with the remaining 5%.**

**Study this document and you will soon be able to control light systems, relays boards, alarms, B&O Masterlink gateway, TVs, screens, projectors, amplifiers,**

**- and much much more.**

# USB SUCKS !!!

Sorry, I do not have too much good to say about USB.
If you are a big fan of USB, then skip this page.
--

USB means Universal Serial Bus, but a better translation would be:

“Useless Serial-communication Bollocks” or “Unstable Serious Bullshit”.

Agreed, it is nice to be able to connect a device directly to the computers USB port, as it holds power as well. I find it useful for updating the GPS navigator, MP3 player, etc. - but that is where the fun stops.

USB is not suited for control purposes, as USB is only valid if a computer is involved, since USB works with a master/slave relationship. Driver and address allocation is required.

If you connect two USB serial-cables using the same type of driver, Windows is most likely not capable of assigning two different comports, but will assign the same comport to both cables.

We often see that an already busy comport is assigned to the serial-cable, and nothing works.

You have interface problems as dedicated cables are needed and if you do not have the right cable, you cannot make yourself one.

Moving a USB cable from one USB port to another, will cause the computer to assign a new comport, but many programs cannot detect which comport was installed and the user have to go through the device manager. Many programs have a hard-coded list of comport numbers and the one selected by the USB driver - sorryyyyyy, is not on the list.

We have seen many computers lock/close the serialport and USB cable stops working.

Devices designed to work with a computer on USB cannot be controlled from a PLC or other of the millions of controllers holding standard RS232, like PLC’s.

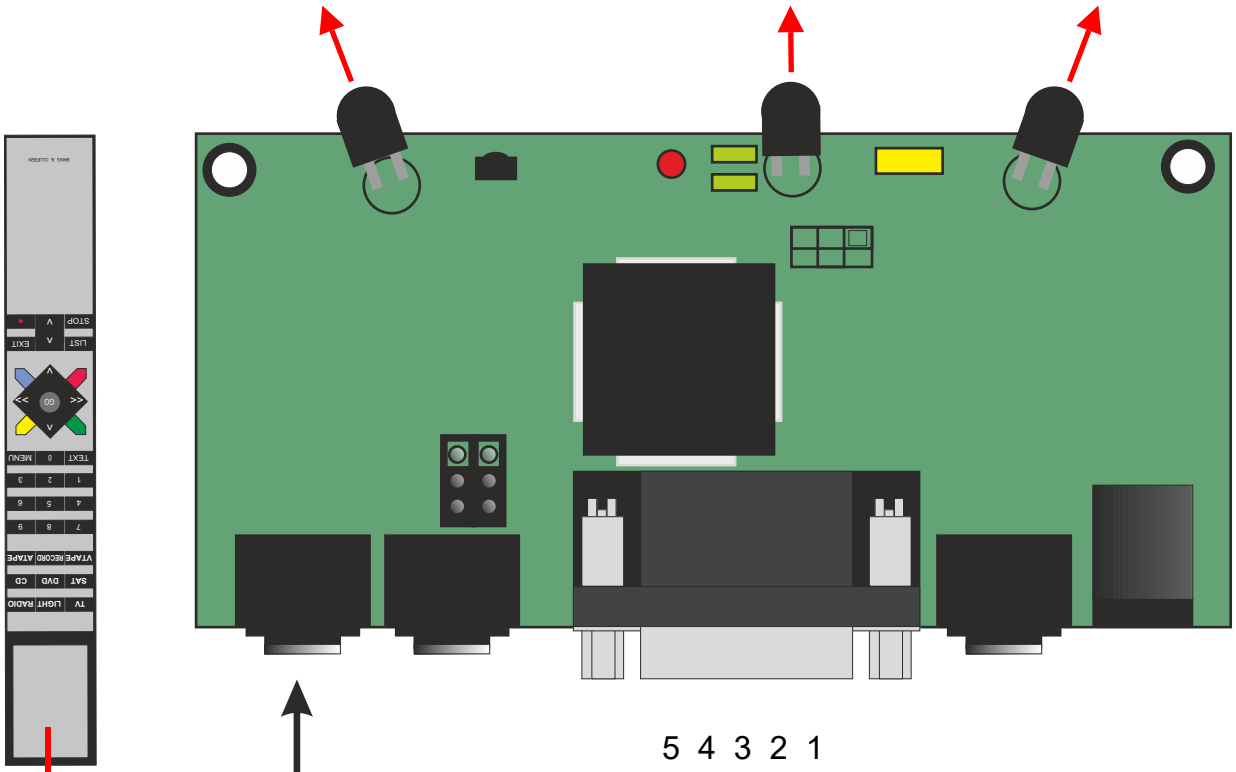
You cannot connect two slave-devices to each other.

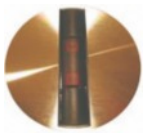
In “the good old days” RS232 was a lot easier to handle and you could connect all kind of devices holding RS232. No master/slave relations and no drivers were involved.

Luckily many manufacturers are still putting old-fashioned RS232 into their products, allowing us to control their products from a wide range of controllers.

This document will show you how easy it is to get RS232 devices to communicate.

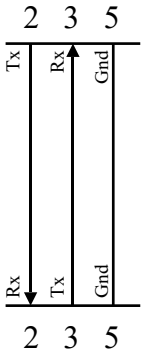
# IR to RS232 - RS232 to IR



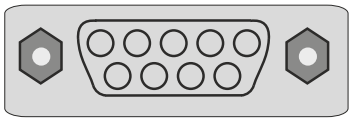


IR RECEIVER

Selfmade cable

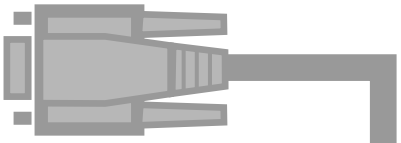



5 4 3 2 1

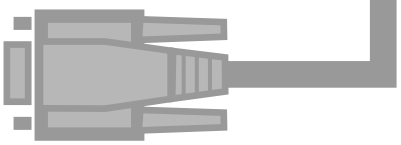


9-pin female

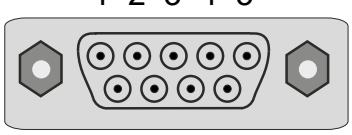
9 8 7 6



ComCab-001



1 2 3 4 5



9-pin male

6 7 8 9

# There are two sites of RS232 ...

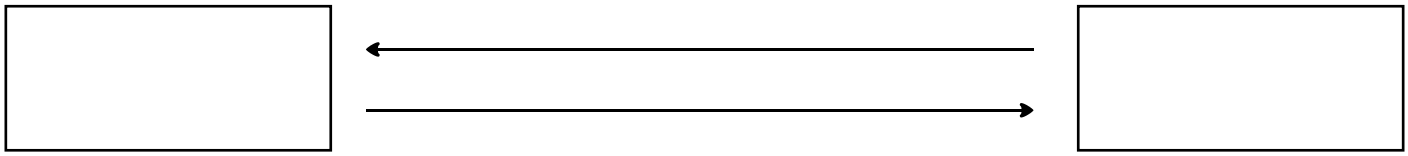
- a complicated site - causing much confusion - even for experts.
- and a simple and easy site - that most people with even limited technical skills can easily learn - quickly.

**The purpose of this document is to enable even non-technicals to get 95% of all RS232 installations to work. The last 5% might need some help from us. You will be able to control light systems, relays boards, B&O Masterlink gateway, TVs, screens, projectors, amplifiers - and much much more.**

We will go deeper into the more technical site of RS232 as we move along - but it might not even be needed.

---

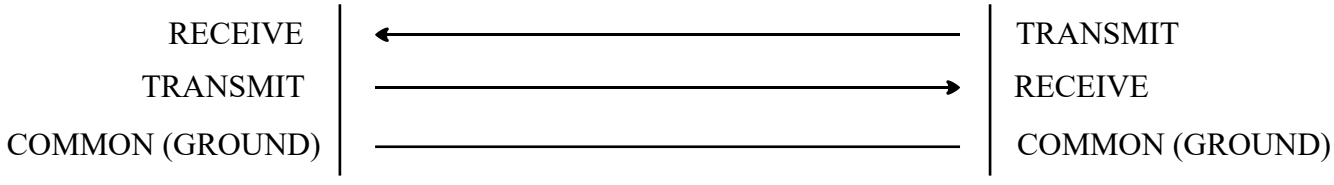
In its basic form an RS232-connection allows two devices to send/receive commands/data to/from each other.



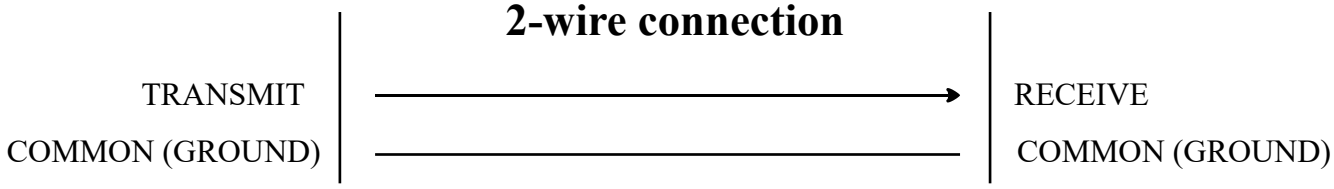
The RS232 protocol and electrical wiring, allows both devices to send and receive commands, but in many of the installations we are involved in, only one device is sending and the other is simply receiving.

The electrical wires needed in most cases, is only 3 wires (sometimes only 2). Normally the type of wiring is not particular critical - but “twisted pair” is recommended. For example CAT5 is of the type “twisted pair”, which means that the wires are paired 2 by 2 and twisted to reduce/avoid noise interference.

**3-wire connection**




**2-wire connection**



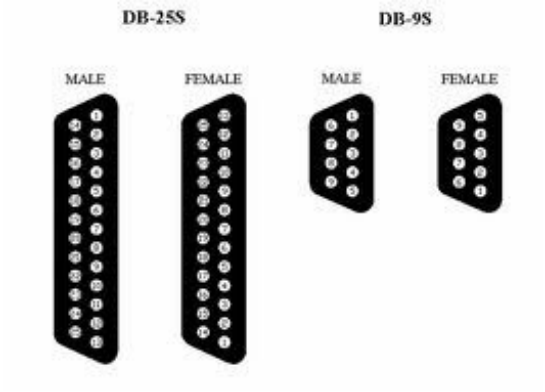
# Connector types

Your device can easily have RS232 available on a connector type you would not expect.

From the very beginning of RS232, a 25-pin connector (DB25) was used to carry a lot of signal wires.

Later, most devices were using 9-pin (DB09) connectors.

Today it is not unusual that devices are using
8-pins connectors (RJ45) or 3-way connectors (Jack).

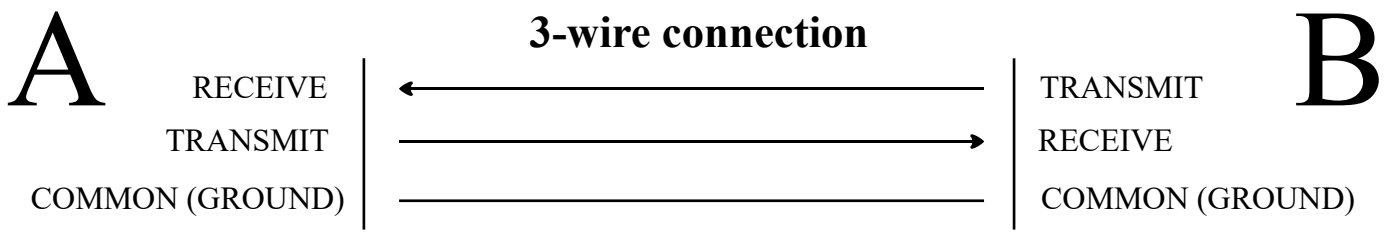



Different manufacturers most likely use different pin-assignment, even if they both use for example a stereo jack connector to carry a 3-wired RS232 signals.

---

It is very easy to connect two devices using 3-wired RS232 communication.

**ALWAYS - connect:**
- Device A's RECEIVE pin to Device B's TRANSMIT pin, and
- Device A's TRANSMIT pin to Device B's RECEIVE pin, and
- Device A's GROUND pin to Device B's GROUND pin.




BUT ... NEVER assume a particular pin-assignment - always check it.

The RECEIVE (RX) signal can easily be pin 2 on device A, but can easily be pin 3 on device B, which does not matter - simply follow the drawing above - ALWAYS.

# Typical connections

An RS232 connector on a computer - even if sitting on a an USB/RS232 converter cable, ALWAYS holds a male connector.

A male connector - when looking into the male pins from the front - ALWAYS count from left to right: 1 - 2 - 3 - 4 - 5
6 - 7 - 8 - 9

On a computer holding a 9-pin RS232 connector:
- pin 2 is ALWAYS RECEIVE
- Pin 3 is ALWAYS TRANSMIT
- Pin 5 is ALWAYS GROUND.

This type of pin-assignment is called DCE (Data Computer Equipment).

---

A device supposed to be connected to a computer, can hold either a male or a female connector. Most typically it will hold a female connector.

A female connector - when looking into the female holes at the front - ALWAYS count from left to right: 5 - 4 - 3 - 2 - 1
9 - 8 - 7 - 6

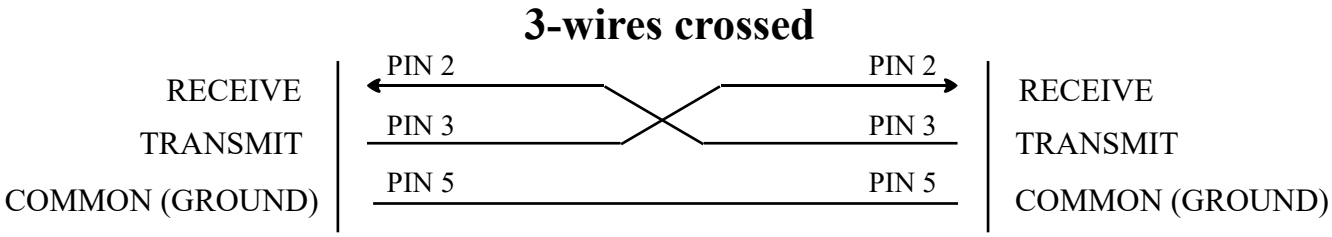
In order to communicate with the computer:
- pin 2 must be TRANSMIT (but could easily be RECEIVE)
- Pin 3 must be RECEIVE (but could easily be TRANSMIT)
- Pin 5 is ALWAYS GROUND.

This pin assignment is called DTE (Data Terminal Equipment).

---

**Please OBSERVE:**

There are NO absolute rules as to whether a relay board, a light system, a projector or anything else for that matter, is to be considered as a DCE or a DTE - whether is should be equipped with a male or a female connector - so always check whether pin 2 or pin 3 is RECEIVE or TRANSMIT. If you connect two systems of the same type (DCE/DTE) then pin 2 and pin 3 must be crossed.



# Detecting the Transmit pin

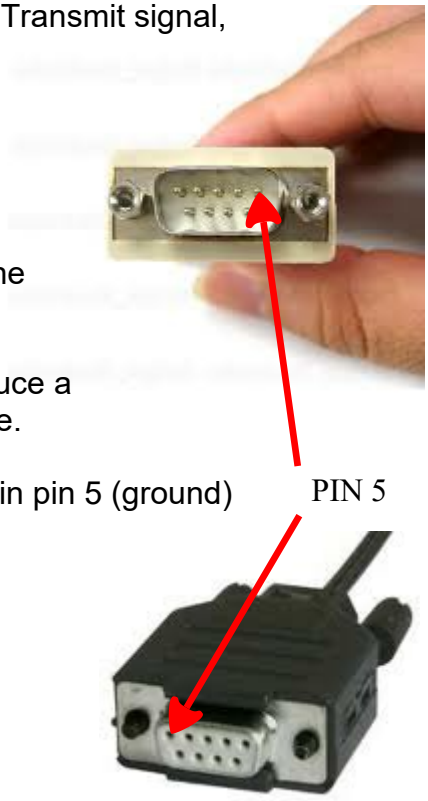
Since you cannot always trust whether pin 2 is the Receive or the Transmit signal, we need to find out, in order to prepare the cabling.

The “trick” I am about to show you, has always worked for me:

Start by the fact that: **Pin 5 is always ground.**

Then all we really need to determine, is whether pin 2 or pin 3 is the Transmit signal. The remaining pin must be the Receive signal.

According to the RS232 specifications, the transmit pin must produce a voltage level of approximately -9 volt to -12 volt when not operative.




In the picture below, I have carefully inserted a bended paper clip in pin 5 (ground) and a bended paperclip in pin 2 of the female RS232 connector.

The volt-meter shows -9 volt.

I can therefore conclude, that on the LinTronic TT455-RT-238, pin number 2 is the Transmit signal.

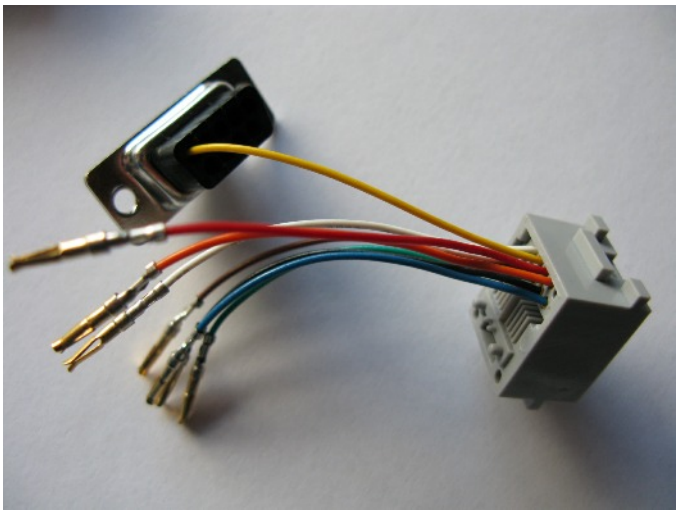
Then pin 3 must be the Receive.

# Connecting ...

An easy way to create customized RS232 cables/connections, is by means of the CAT5-RS232 converters.

They allow you to insert the wires in the RS232 connector in any way you like and use a CAT5 cable to connect the two RS232 devices.

No tools are required.



If you insert the wires incorrectly, no harm is done. In worst case it simply will not work - but can be corrected. To remove a wire you will need a little special tool.

Or you can solder wires as shown below.

If you make a mistake and connect pin 2 and pin 3 incorrectly - it simply will not work - but can easily be corrected.

# Baudrate, Parity, Databits, Stopbits

When the wiring is in place, all you need to ensure is that:
- **these 4 settings must be identical on the two devices which should communicate.**

Normally these settings can be adjusted, and some times they are fixed on one of the devices.

The **baudrate** is the communication speed. It is expressed as “bits pr seconds” (bps) and normally is a number between 2.400 bps (slow) and 115.200 bps (fast).

Longer wires require a slower communication speed in order to work properly.
Shorter wires can operate on higher speeds.

Baudrates that normally will work perfectly in any environment is 9.600 bps, 19.200 bps or 38.400 bps.

We recommend 19.200 bps.

**Parity** is a way for both devices to check for communication errors. It is expressed as None, Even og Odd.
Normally it is set to None.

**Databits** is an indicator for how many bits are used to specify one character.
Normally it is set to 8.

**Stopbits** is an indicator for how much time is required by the receiving device to handle the incoming data.
Normally it is set to 1.

---

# Basically that’s it ...

You are now ready to rock and roll.

With this basic information you are now able to connect our TT455-RT-238 Universal Signal Converter or our RELAY 208, to for example a light system, a projector - or any other device holding RS232.

If you want to know more of the technical stuff, then consult the following pages.

# Commands / protocol

Any RS232 device is controlled by a set of commands.

A command is an instruction telling the device what to do - or asking the device for information.
The commands (or the replies) can be very simple (only 2 or 3 characters) or they can be very long and hold all sorts of data (like for example the date and time, a temperature, etc. etc.)

An example of a command could be:
- PON (asking my projector to turn the power On)
- POF (asking my projector to turn the power Off)

In other words - in this case - we only need to send 3 characters - P O N - in order to turn the power On.

---

**More complex communication - often called a protocol**

Other devices use a more complex syntax (many characters), but are still easy to understand.

The LinTronic command set (also known as our protocol) follows a specific pattern, which allows us to send a specific command with specific data to a specific device:

< T T F F C C C D D D D . . . . . X X X >

< = start of command (also called start of package, or start of telegram)
T T = a 2-digit instruction specifying the number/address of the devices - supposed to receive the package
F F = a 2-digit instruction specifying the number/address of the devices - that send the command
C C C = a 3-digit instruction specifying a Command - the reason for the package - what to do
D D D = data to be used by the receiving device (data are different for different commands)
. . . = more data
X X X = a 3-digit checksum - enables the receiving device to check that what was send was also received
> = end of command (also called end of package, or end of telegram)

In other words:
The receiving device will only accept the package:
- if it starts with a < and ends with a >
- if the TT number matches the device address/number
- if the Command is a known command
- if data makes sense
- if the checksum is correct

All this just means that we can send a complex instruction which is only carried out if everything is ok.

---

This was only a few examples to show you what you can expect when you go looking for command sets.
For even more technical insight - keep reading :o)

# Commands are send one character at a time - like a morse code

All in all, you can send up to 256 **different** characters. These 256 characters are for example the entire alphabet (either small letters like a-z or capital letters like A-Z, but also the numbers 0-9 and signs like for example ! “ # ¤ % & / ( ) , ; : .

All of these characters you already know from your keyboard. But the computer also works with a number of invisible characters not shown on your keyboard.

For a example, the ENTER key is really a combination of two control characters:  CarriageReturn and LineFeed.

In the old days, the CarriageReturn was commanding the typewriter or the mechanical telex machine to return the printing head to the beginning of the paper, and the LineFeed commanded the paper to be advanced one line. None of these characters are shown on print, but only controlling the machine in order for the text to appear as wanted.

A wide range of other invisible control characters are available for specific purposes: TAB, StartOfText, EndOfText, XON, XOFF, etc.

---

Earlier we mentioned a very simple command like: PON

A command is send as individual characters. In this case: First P, then O and finally N.

They are send right after each other, as fast as the computers resources allow it - at a speed known by both the transmitting and the receiving device.

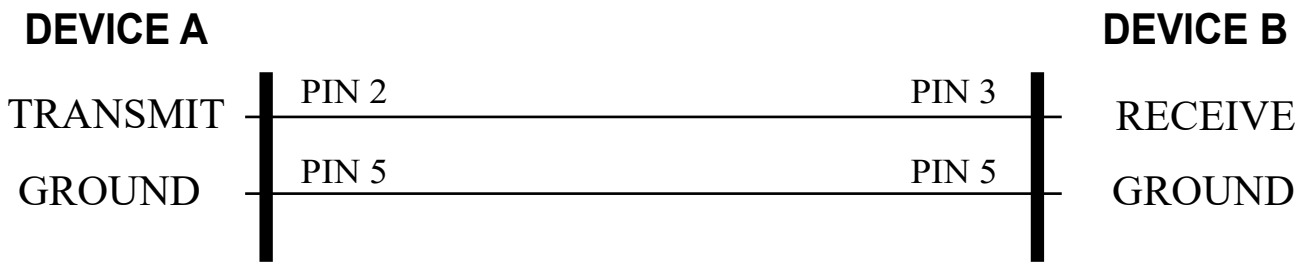
As a very simple example - you could say that a small lamp is blinking - like a morse code - and that the receiving device is translating the timelength of the blinks into the relevant characters.

In an electrical wire, the blinks are represented as a voltage level:
- Minus 9 volt is no blink (lamp Off)
- Plus 9 volt is a blink (lamp On)

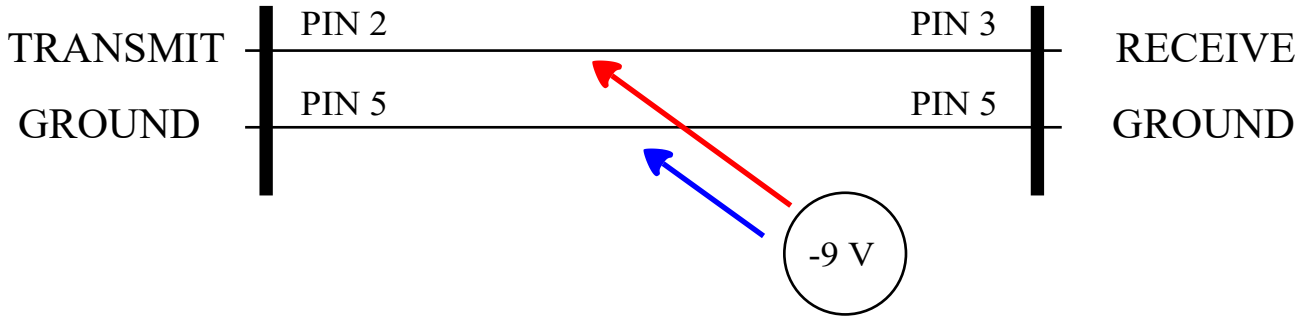
On the next page we will show you how we send 256 different characters and how the receiving device translates the electrical signals into characters.

You will see how the three characters P, O and N (and any one of the 256 characters) follow the exact same pattern.

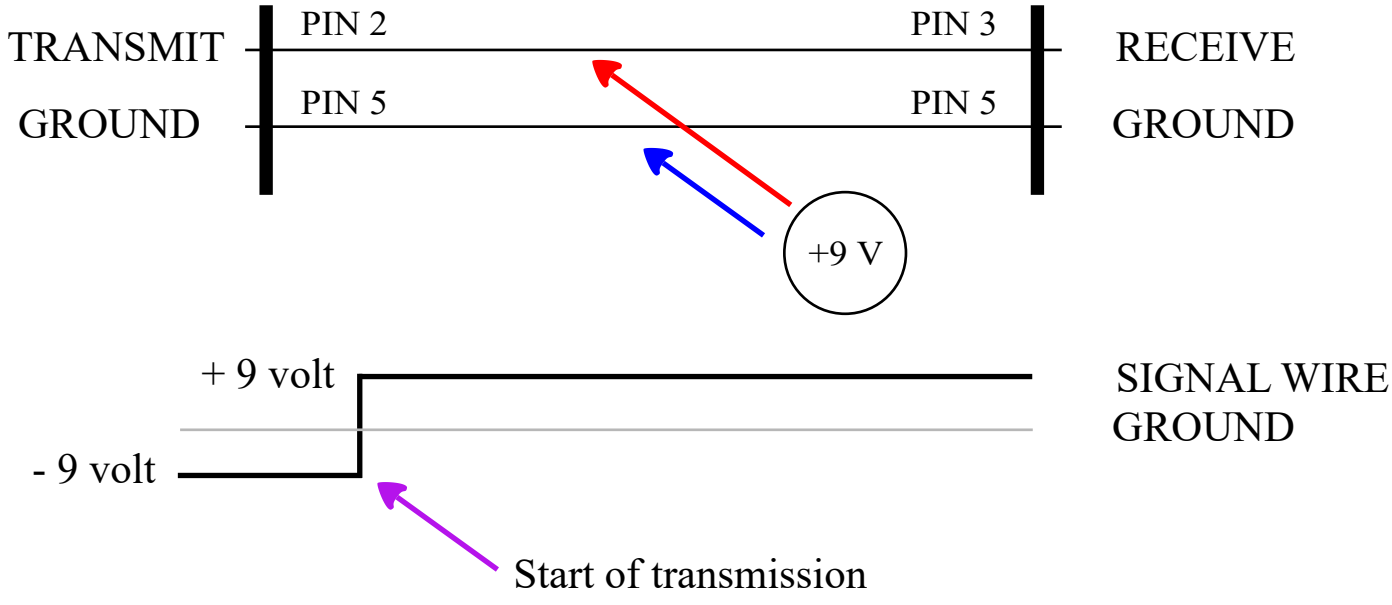
In this example, we will only send data one way. Sending data the other way is 100% identical.



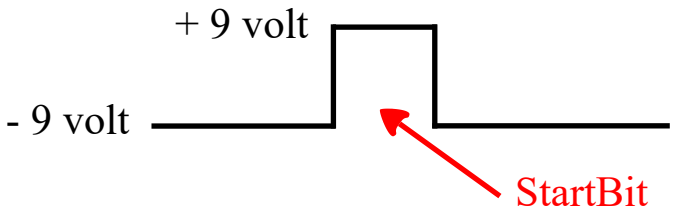

**IDLE - NO DATA**
When Device A is not sending data to Device B, the voltage level between pin 2 and pin 5 is Minus (-) 9 volt.




**START OF TRANSMISSION**
When Device A wants to start sending, it will raise the voltage level from Minus (-) 9 volt to Plus (+) 9 volt.
Device B will detect that the voltage level changes and will make itself ready to start receiving data.



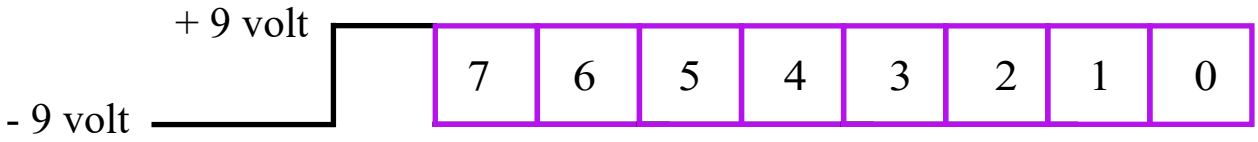
**START BIT**
The timelength of the startbit is 1 second divided by the baudrate. The higher the baudrate/speed is, the shorter is this time reference. All time periodes is of the same timelength as the startbit. StartBit is always On (1).



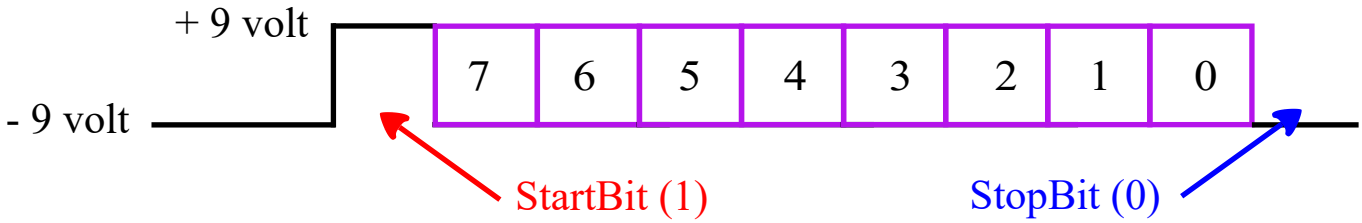

**DATA BITS**
After the StartBit, Device B is fully ready to receive data, and device A will send 8 databits, which means that it will change the voltage level 8 times - from Minus 9 volt to Plus 9 volt - or from Plus 9 volt to Minus 9 volt, depending on the “morse code blink-pattern” required to signal the particular character.

We call Plus 9 volt for On (1) - and Minus 9 volt for Off (0). 8 combinations of On/Off gives a total of 256 combinations.

The Databits are numbered from 7 to 0. We send number 7 as the first and number 0 as the last.




**STOP BIT**
After the Databits, we insert a stop bit. A stop bit is a pause - a break. It allows the receiving Device B, to handle the received data and find out what to do with it. When the stop bit has expired (same time period) Device A can continue sending the next characters. StopBit is always Off (0).




**TIMING**
Including a startbit, 8 databits and a stopbit - one character is 10 bits.
The time it takes to send 1 complete 10-bit character is ( ( 1 second / baudrate ) * 10 ).
At a baudrate of 9600 bps it will take ( 1 / 9600 ) * 10 ~ 0,001 second ~ 1.04 ms.

**256 CHARACTERS**
8 combinations of On (1) and Off (0) , gives us a total of 256 characters: From character 0 to character 255).

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

If the location is represented by a 0, then you do not add the related number.
If the location is represented by a 1 you add the related number.

| 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | | 7 6 5 4 3 2 1 0 |
|---|---|---|---|
| 0 0 0 0 0 0 0 0 = 0 | 0 0 0 0 1 0 0 0 = 8 | ..... | 1 1 1 1 1 0 0 0 = 248 |
| 0 0 0 0 0 0 0 1 = 1 | 0 0 0 0 1 0 0 1 = 9 | | 1 1 1 1 1 0 0 1 = 249 |
| 0 0 0 0 0 0 1 0 = 2 | 0 0 0 0 1 0 1 0 = 10 | | 1 1 1 1 1 0 1 0 = 250 |
| 0 0 0 0 0 0 1 1 = 3 | 0 0 0 0 1 0 1 1 = 11 | | 1 1 1 1 1 0 1 1 = 251 |
| 0 0 0 0 0 1 0 0 = 4 | 0 0 0 0 1 1 0 0 = 12 | | 1 1 1 1 1 1 0 0 = 252 |
| 0 0 0 0 0 1 0 1 = 5 | 0 0 0 0 1 1 0 1 = 13 | | 1 1 1 1 1 1 0 1 = 253 |
| 0 0 0 0 0 1 1 0 = 6 | 0 0 0 0 1 1 1 0 = 14 | | 1 1 1 1 1 1 1 0 = 254 |
| 0 0 0 0 0 1 1 1 = 7 | 0 0 0 0 1 1 1 1 = 15 ( 8+4+2+1) | | 1 1 1 1 1 1 1 1 = 255 |

All what is left now, is to allocate all of the different characters from the keyboard to individual numbers.

There is an international standard for that, called the ASCII table.
Please see: http://www.asciitable.com

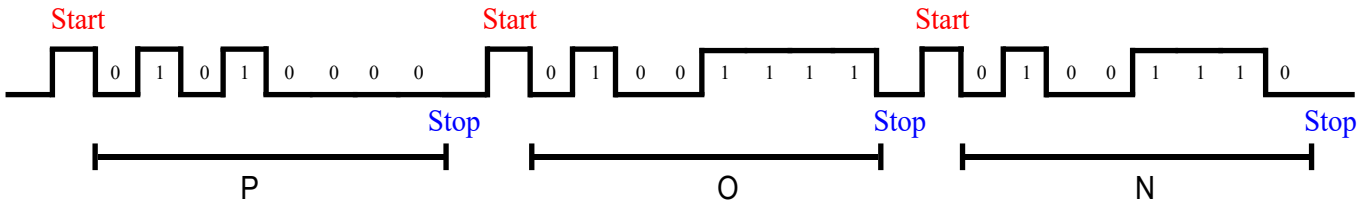
Some examples:

| | | |
|---|---|---|
| - A LineFeed is known as character 10. | 10 = 8 + 2 | = 00001010 |
| - A CarriageReturn is known as character 13. | 13 = 8 + 4 + 1 | = 00001101 |
| - A Space is known as character 32. | 32 = 32 | = 00100000 |
| - The digit 0 is known as number 48. | 48 = 32 + 16 | = 00110000 |
| ... | | |
| - The digit 9 is known as number 57. | 57 = 32 + 16 + 8 + 1 | = 00111001 |

In order to send PON we must send character number 80 (P), number 79 (O) and number 78 (N).

| | | | |
|---|---|---|---|
| P | = 80 | = 64 + 16 | = 01010000 |
| O | = 79 | = 64 + 8 + 4 + 2 + 1 | = 01001111 |
| N | = 78 | = 64 + 8 + 4 + 2 | = 01001110 |

It would look like this:



# Clear text, Ascii or HEX ?

This section explains one of the most confusing matters, when discussing RS232 control.

On the previous pages you saw that an 8-bit combination of 0's and 1's make a total of 256 characters, from 00000000 (0) to 11111111 (255).

Of these 256 characters the characters with a number lower than 33 are invisible. The first 31 characters are completely invisible. Number 32 is the space bar, which you can only see if something else surrounds it.

It is very common that an RS232 command is terminated by a CarriageReturn, but you have to describe that in clear text, as you cannot type the CarriageReturn.

**PS: In the below have terminated the PON command with a CarriageReturn.**

We want to be able to present all characters, including the invisible and region-related characters - but how do you specify/type an invisible character ?

We could type all characters as their 8-bit combination, but it would take up a lot of space.
PON in 8-digit binary = 01010000 01001111 01001110 00001101
P O N CarriageReturn

We could type all characters as their 3-digit ASCII value, which is shorter:
PON in 3-digit ASCII = 080 079 078 013
P O N CarriageReturn

Finally, we could type all characters as their 2-digit HEX value, which is even shorter:
PON in HEX = 50 4F 4E 0D
P O N CarriageReturn

Due to the small space required, and the fact that all 256 commands can be typed, HEX looks interesting.

So how do we convert the 8-bit combination into HEX ?

# HEX - basics

We convert into HEX, by splitting the 8-bit combination into two 4-bit combinations.

The four 0 and 1 locations are then assigned a value:

| 8 | 4 | 2 | 1 |
|---|---|---|---|

If the location is represented by a 0, then you do not add the related number.
If the location is represented by a 1 you add the related number.

| 8 4 2 1 | 8 4 2 1 | 8 4 2 1 | 8 4 2 1 |
|---|---|---|---|
| 0 0 0 0 = 0 | 0 1 0 0 = 4 | 1 0 0 0 = 8 | 1 1 0 0 = C |
| 0 0 0 1 = 1 | 0 1 0 1 = 5 | 1 0 0 1 = 9 | 1 1 0 1 = D |
| 0 0 1 0 = 2 | 0 1 1 0 = 6 | 1 0 1 0 = A | 1 1 1 0 = E |
| 0 0 1 1 = 3 | 0 1 1 1 = 7 | 1 0 1 1 = B | 1 1 1 1 = F |

The 5 characters A, B, C, D, E and F are introduced in order to keep the amount of characters needed to only 1.

In order to start softly, we will now convert the command PON, from 8-bit combinations into HEX.

We start with the letter P:

The ascii number for P is 80. In binary code, 80 is: 01010000

Splitting it into two 4-bit combinations, you get: 0101 and 0000

Looking into the table above, we translate 0101 into 5 and 0000 into 0.

The HEX code for P is then 50.

Following the same logic, you will find that the letter O (ascii 79), translates into the 8-bit combination 01001111 which is split into 0100 and 1111 which becomes 4F.

Following the same logic, you will find that the letter N (ascii 78), translates into the 8-bit combination 01001110 which is split into 0100 and 1110 which becomes 4E.

If the command is terminated by CarriageReturn (ascii 13), that translates into 0D in HEX.

256 ASCII characters from 0 to 255, translates into 256 HEX characters from 00 to FF.

# HEX - and why it is so confusing

When we have converted an ascii character with a a number ranging from o to 255, then we can represent these characters with a 2-digit HEX format.

Many technicians and even programmers believe that ASCII and HEX characters are different types of character sets.

THEY ARE NOT. There is only ONE set of characters.

Find them here: http://www.asciitable.com

Note that for example the letter P has a decimal value (80) and a HEX value (50), but the electrical signal for P when send on the RS232 connection is exactly the same:

| Letter | ascii/decimal | 8-bit/binary | HEX | | | | |
|---|---|---|---|---|---|---|---|
| P | 80 | 01010000 | 50 | 0101 | + | 0000 | = 01010000 |
| | | | | 5 x 16 | + | 0 | = 80 |

**CONFUSING**

If you are new to RS232 (comport communication) and start sending out commands, then be prepared for spending some time on finding out what is correct.

HEX formatted commands often listed in manuals as 0x50, 50x, 50H or simply 50.

Very often we read in manuals that the commands must be send out as HEX commands.

Sometimes that simply means that the command P should be translated from HEX (50) to the ascii number (80) and send out as any other ascii character in its binary code 01010000.

But some times it means that the HEX command must be send out exactly as it is printed.

In other words HEX command 50 must be send out as two keyboard numbers: 5 (five) and 0 (zero).

But keyboard number 5 is ascii character 53 and keyboard number 0 is ascii character 48.

Isn't that simply a matter of sending clear text ? Yes, it is.

Clear text commands are often referred to as ascii commands, but ascii is all of the 256 character, not only the visible keyboard characters.

# Handshake signals

If needed, this document will later include explanation about handshake signals.

Until that problem arise, we will keep it out of this document.