

RS232 ... What is it ?

It is a method (or protocol - an agreed standard) that defines how to transfer data between two devices using a few wires. It uses a serial transmission method where bytes of data are output one bit at a time onto a single wire.

Data is only transmitted in **one direction for each wire** so for bi-directional communication (two directions) you need **two wires**.

These two along with a ground reference (total: three wires) make up the minimum configuration that you can get away with.

Note: For more reliable communication over long distances you may need more wires for handshake signals etc.

More formally RS232 is an asynchronous communication protocol that lets you transfer data between electronic devices.

Basically it can transfer a single byte of data over a serial cable having between 3 to 22 signals and running at speeds from 100 to 20k baud. Common baud rates used are 2.4k, 9.6k, 19.2k, The cable length can be up to 50ft. Higher baud rates are used but not covered by the standard they still work though e.g. 38400,57600 Baud (bits/s).

To transfer a block of data individual bytes are transmitted one after another.

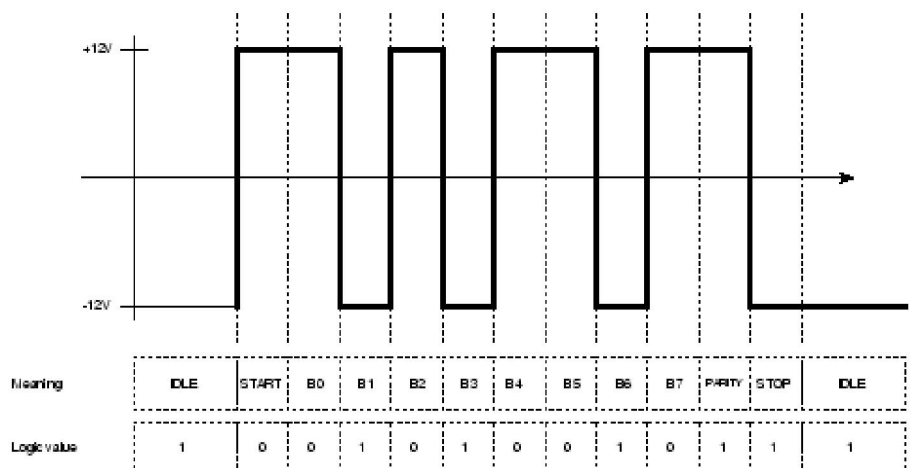
How RS232 works

This section describes how RS232 works in general without describing complex handshake methods - only the simplest system is described - this is the most **useful** and the **most likely to work!**

Data is transmitted serially in one direction over a pair of wires. Data going out is labeled Tx (indicating transmission) while data coming in is labeled Rx (indicating reception). To create a two way communication system a minimum of three wires are needed Tx, Rx and GND (ground). Crossing over Tx & Rx between the two systems lets each unit talk to the opposite one.

Each byte can be transmitted at any time (as long as the previous byte has been transmitted). The transmitted byte is not synchronized to the receiver - it is an asynchronous protocol i.e. there is no clock signal. For this reason software at each end of the communication link must be set up exactly the same so that each serial decoder chip can decode the serial data stream.

RS232 Transmission of the letter 'U'



Note: The signal level inversion (logic 1 is -12V and logic 0 is +12V).

How RS232 works : Baud

This is simply the transmission speed measured in bits per second. It defines the frequency of each bit period.

For a baud rate of 2400 (2400 bps) the frequency is 2400Hz and the bit period is 1/2400 or 416.6us. This is the information that a receiver uses to recover the bits from the data stream.

How RS232 works : Voltage levels

Transmitter

How RS232 works when transmitting a data bit stream.

To make it work over long cables high voltages are sent from each transmitter since due to cable resistance the voltage reduces the further the signal has to travel.

The output voltage specification is from +5V to +25V (transmitting a logical zero) and -5V to -25V (transmitting a logical one).

Note: all signals in the cable have to generate the same voltage levels e.g. DTR, DSR, RTS, CTS. So you need a lot of level translator chips for a full interface but for very short distances you only need TX and RX and ground.

The receiver can accept minimum signal levels of $\pm 3V$.

The maximum voltage of $\pm 25V$ does not have to be used and a common voltage in use is $\pm 12V$ (output by MAX232 transceiver chip).

A mark (logical one) is sent as -12V and a space (logical zero) is sent as +12V i.e. the logic sense is inverted.

Note: The fact that high voltages exist at the serial port allows powering devices that you would not normally expect to find on it. But they must draw very little current.

Receiver

How RS232 works when receiving a data bit stream.

At the receiver the input voltage levels are defined as $\pm 3V$ i.e. to receive a logic zero the voltage must be greater than 3V and to receive a logic one the voltage must be smaller than -3V.

This allows for losses as the signal travels down the cable and provides noise immunity i.e. any spurious noise up to a level of $\pm 3V$ can be tolerated without it having any effect on the receiver.

How RS232 works : Bit stream

How RS232 works - the Start Bit

The protocol is described as asynchronous as there is no clock transmitted at all. Instead a different method of clock recovery is used.

At the beginning of each transmission a start bit is transmitted indicating to the receiver that a byte of data is about to follow.

The start bit lets the receiver synchronize to the data bits. What this means is that the receiver can create its own sample clock at the middle of each bit. Note that once the start bit is found the receiver knows where the following bits will be as it is given the sample period (derived from the baud rate) as part of the initialization process.

How RS232 works - the Data bits

Data bits follow the start bit. There will be seven or eight data bits with the lsb transmitted first. The reason you can choose between seven or eight is that ASCII is made up of the alphabet within the first seven bits (as well as the control characters). The eighth bit extends the character set for graphical symbols.

If you only want to transmit text then you only need 7 bits. This saves a bit and increases transmission

speed when transmitting large blocks of data. Other data bit sizes are 5 and 6 bits. However bit length is usually ignored and a transmission size of 8 bits is commonly used.

Note: If you use RS232 to transmit raw data (binary data) then you will need 8 data bits.

How RS232 works - the Parity Bit

The parity bit is a crude error detection mechanism. You can use either odd parity or even parity or none at all (in this case no parity bit is transmitted).

It simply evaluates all the data bits and for odd parity returns a logic one if there is an odd number of data bits that are set. For even parity an even number of data bits that are set, sets the parity bit.

At the receiver the parity bit is used to tell if an error occurred during transmission. You can use this in the receiver software by reading a flag in the UART module.

The problem with error detection using the parity bit is that if two bits are in error then the parity check fails. This is because each error cancels the effect of the other (in terms of the parity calculation). Any even number of errors causes a failure in error detection.

It won't be a problem on a bench top based system (that has no critical data transfer). Over a short cable e.g. 6ft you probably won't see any errors anyway. Normally I use no parity and there is no problem at all.

For systems running over a long distance or in a noisy environment a better system should be used e.g. Adding a cyclic redundancy check to the data stream before and after it is sent ver the RS232. CRCs let you check for and correct quite a few errors without re transmitting the data.

How RS232 works - the Stop bit

The stop bit merely gives a period of time before the next start bit can be transmitted. It is the opposite sense to the start bit and because of this allows the start bit to be seen.

If there was no stop bit then the last bit in the data stream would be the parity bit (or data bit if parity is not active). This would change depending on the data sent so if it had the same sense as the start bit then the start bit could not be seen!

The stop bit can be set choosing from 1, 1.5, or 2 bit periods.

How RS232 works : Typical Settings

Typical settings for use on the desktop e.g. between a microcontroller and hyperterminal:

How RS232 works : Typical Baud rate settings.

Baud	9600
Data bits	8
Parity	None
Stop bits	1

Hardware Connections 3 (Rx,Tx,GND) - Rx and Tx crossed over.

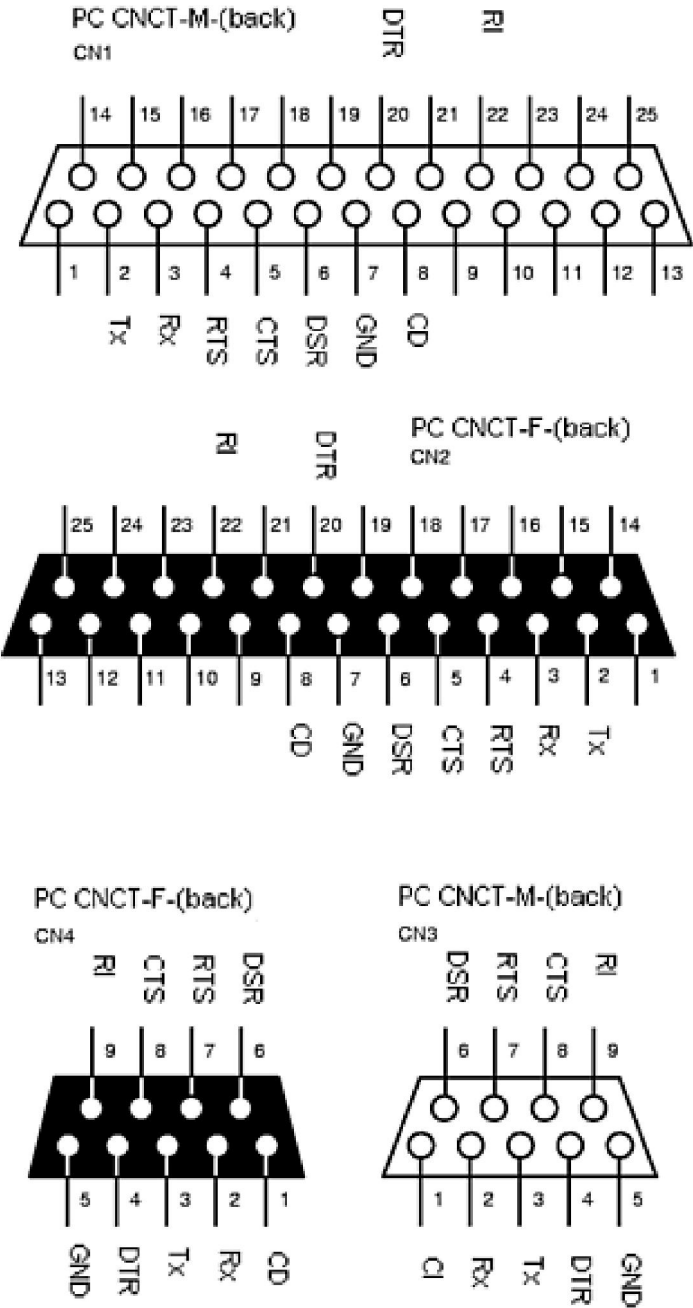
Signals : How RS232 works with signal levels

At some point you may want to make a software UART perhaps to save code space in your current design (maybe you don't need the receive part - just outputting variables) or to use a spare pin or perhaps your provider's library does not work.

To create it you need the actual signal diagrams that you see at the microcontroller pin (strangely these are hard to find on the web).

The following diagram shows how RS232 works by generating 0-5V logic bit stream at the output pin of the microcontroller or UART followed by the translated voltages that are transmitted to the serial

cable. These are generated by sending the 0-5V logic levels to a transceiver chip e.g. MAX232. which can use a 5V power supply and boost it to the required 12 volts.



End of page : How RS232 works.