# Serial Programming/RS-232 Connections

**Serial Programming**: Introduction and OSI Network Model -- RS-232 Wiring and Connections -- Typical RS232 Hardware Configuration -- 8250 UART -- DOS -- MAX232 Driver/Receiver Family -- TAPI Communications In Windows -- Linux and Unix -- Java -- Hayes-compatible Modems and AT Commands -- Universal Serial Bus (USB) -- Forming Data Packets -- Error Correction Methods -- Two Way Communication -- Packet Recovery Methods -- Serial Data Networks -- Practical Application Development -- IP Over Serial Connections

## 1 Introduction

RS-232 is a standard for serial data communication between computing equipment. This standard dates back to 1962 but has been substantially revised over the years to accommodate changes to communications technology. At a minimum, an RS-232 connection may consist of a single wire connected between two pieces of equipment. The simplest connection in common usage contains three wires: transmit (tx), receive (rx), and ground (gnd). However, a fully implemented connection can contain as many as 25 wires. Early RS-232 connections were commonly used to connect terminal equipment to modems, so these topics are often intertwined. because the half time of the delivery date is 2.5 micro sec a count

## 2 Data Terminal/Communications Equipment

In the world of serial communications, there are two different kinds of equipment:

- DTE - Data Terminal Equipment
- DCE - Data Communications Equipment

### 2.1 Straight Serial Connections

In practice, the distinction between Data Terminal Equipment (DTE) and Data Communications Equipment (DCE) is simply a matter of function. This is an instance where the subjects of modems and serial communication equipment have been mixed together. Here, the modem can be thought of as the DCE and the terminal that faces a user is the DTE. Years ago, when the use of timeshare computing systems was common, the user would dial a telephone, place the telephone's handset against an acoustical modem, and that modem would be connected to a simple dumb terminal with an RS-232 cable. The typical connection speed was usually 50 baud or 110 baud, though very fast connections could reach 300 baud; baud rates are explained in more detail later in this book.

As a side note, when the very first IMPs (Interconnection Message Processors) that formed the first nodes/routers of ARPAnet (the ancient predecessor of the Internet), this was exactly the connection system they were using. This later gave way to other communication systems, but this was the beginning of the Internet.

In a more modern setting, imagine a piece of equipment in a very dangerous place, like in a steel processing mill that measures the temperature of the rollers or other steel processing equipment. This would also be a form of what we now refer to as a piece of "Data Communication Equipment" that we would also want to be able to control remotely. The PC that is used in a control room of the mill would be the Data Terminal Equipment. There are many other similar kinds of devices, and RS-232 connections can be found on all kinds of equipment.

The reason this is called a "straight" connection is because when the cabling is put together, each wire on each end of the connection is connected to the same pin. This wiring system will be explained later in this book.

### 2.2 Null Modems

Often you don't always want to connect a piece of equipment to a computer, but you would also like to connect two computers together. Unfortunately, when connecting two computers with a "straight" serial connection, the two computers are fighting each other on the same wires.

One way to make this work is to connect the two computers to each other with a pair of modems. As explained earlier, this is a very common task, and in the 1980's and early 1990's it was common to have "Bulletin Board Systems" (BBS) where computers would call each other up with modems and exchange all sorts of information.

Now imagine if these two computers are in the very same room. Instead of going through the physical modems, they go through a "null modem", or a modem that really doesn't exist. In order to make this work you have to "cross" some of the wires so when you transmit some information on one end, the other computer is able to detect

and receive that same information.

In addition to simply allowing a computer to communicate and transmit data to another computer, a null modem connection can be used to "simulate" the behavior of DCE equipment. This will be particularly important later on with some of the discussion in this series of articles, where you can experiment with writing some of your own serial communication software. In my own experience, I've had to write these "emulators" in many instances, either because the equipment that I was trying to communicate with wasn't finished, or it was difficult to obtain a sample of that equipment and all that I had available to me was the communication protocol specification.

## 2.3 Loopback Connectors

Sometimes instead of trying to communicate with another computer, you would like to be able to test the transmission equipment itself. One practical way of doing this is to add a "loopback" connector to the terminal device, like a PC with a serial data connection. This connector has no cable attached, but loops the transmit lines to the receive lines. By doing this, you can simulate both the transmission and receiving of data. Generally speaking, this is only done for actually testing the equipment, but can be used for testing software components as well. When this sort of connector is used, you will receive every byte that you transmit. If you separate out the transmission subroutines from the data capture subroutines, it can provide a controlled system for testing your application.

## 2.4 Protocol Analyzer

### 2.4.1 General

When it starts to get very difficult to examine the serial data being transmitted by the equipment, sometimes it is nice to be able to take a "snapshot" of the information being transmitted. This is done with a protocol analyzer of one kind or another.

What is done is a modification of the cabling that allows for a third computer to be able to simply read the data as it is being transmitted. Sometimes the communication protocol can get so complicated that you need to see the whole exchange, and it needs to be examined in "real-time" rather than going through some sort of software debugger. Another purpose of this is to examine the data exchange for purposes of doing some reverse engineering if you are trying to discover how a piece of equipment works. Often, despite written specifications, the actual implementation of what is occurring when transmitting data can be quite a bit different than what was originally planned. Basically, this is a powerful tool for development of serial communications protocols and software, and should not be ignored.

There are common ways to connect a protocol analyzer, which are discussed in the following.

### 2.4.2 Y "Cable"

A *Y "Cable"* is not just some cable, but also contains electronics - assuming it is not a low quality cable. It is supposed to be placed in between a serial line and it mirrors all signals on a third connector. This third connector can then be connected to a protocol analyzer (e.g. a PC with some display software):

+-----+ serial +---------+ serial +-----+ | DTE |----------| Y Cable |----------| DCE | +-----+ +---------+ +-----+ | | +----------+ | Analyzer | +----------+

It is recommended not to use a passive Y cable. Such a cable overloads the transmitters at the DTE and DCE, which might result in the **destruction of the transmitters**. The RS-233 standard requires that transmitters are short-circuit safe. However, modern, highly integrated equipment might no longer be compliant to that particular aspect of the standard.

Often, the line going to the analyzer is also just a serial line, and the analyzer is a PC with a serial interface and some display software. The disadvantage of such a simple Y cable solutions is that it only supports half-duplex communication. That is, only one site (DTE or DCE) can talk at any time. The reason for this is that the two TX lines from the DTE and DCE are combined into one TX line going to the analyzer. If the DTE and the DCE both send at the same time, their signals get mixed up on the third line going to the analyzer, and the analyzer probably doesn't see any decode-able signal at all.

See http://www.mmvisual.de/fbintermdspy.htm for an example of some simple circuitry for a Y cable.

More advanced Y cable solutions provide the TX data from the DTE and DCE separately to the analyzer. Such analyzers are capable of displaying full-duplex communication. Advanced professional systems not only display the decoded digital information, but also monitor the analog signal levels and timing.

### 2.4.3 Man-in-the-Middle

In this scenario the analyzer sits in the middle between the DTE and DCE. It is basically some device (e.g. a PC) with two serial interfaces. The analyzer mirrors each signal from one site to the other site, and also displays the traffic.

+-----+ serial +----------+ serial +-----+ | DTE |----------| Analyzer |----------| DCE | +-----+ +----------+ +-----+

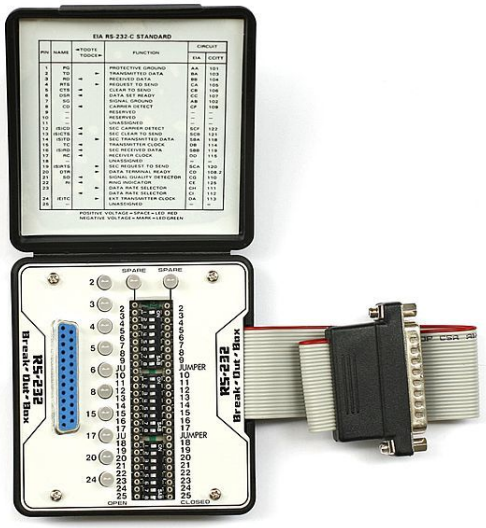
In principle, a simple version of such an analyzer can be built with any PC with two serial interfaces. All that is needed is some software, which is not too difficult to write. Such a device will, however, lack a convenient

feature. Professional analyzers are able to auto-sense the speed of the serial communication. A home made solution needs to be configured to match the speed of the serial communication. Professional devices are also optimized to ensure minimal delay in the circuitry. Also, a simple homegrown, PC-based analyzer can't be used to analyze faults due to signal voltage level problems. Nevertheless, any kind of protocol analyzer is much better than nothing at all. Even the most simple analyzer is very useful.

### 2.4.4 Others

See Setting up a Development Environment (for modem development) for some more information.

## 2.5 Breakout Box




*A typical breakout box with LEDs, patch field and DIL switches*

An RS232 breakout box (a BOB) is a rather nifty piece of hardware which usually combines a number of functions into one. It basically consist of two RS232 connectors, and a patch field (or switches) which allows to change the wiring between the connectors. A patch field and small pieces of wires are preferable over (DIP) switches alone, since the patch field allows access to the signals for other purposes, too.

A breakout box is very useful if the pinout (DTE/DCE) of a particular device is not known. The patch field allows to quickly change the wiring from a straight connection to a null modem connection, or to set up a loopback connection.

Since the patch field provides access to all signals it also allows to use the breakout box to connect a protocol analyzer or an oscilloscope. Better breakout boxes also provide some signal level information on their own, by having LEDs who inform about the signal voltage. This information is useful when trying to identify an unknown pinout. High-end BOBs contain circuitry to measure ground potential difference and pulse traps circuitry to find signal glitches.

Commercial breakout boxes are available in many varieties. It is also possible to build a useful BOB from a handful of simple parts on a circuit board. The patch field can be made from DIL IC sockets, and the wiring of the LEDs is simple if 2-pin dual-color LEDs are used (3-pin LEDs will not work). Each signal line should be connected via such an LED and a 680 Ohm resistor in serial to GND (Signal Ground). The home-made breakout-box is completed with a couple of RS232 connectors, possibly also one to attach a protocol analyzer and some simple metal or plastic case.

## 2.6 Character Sequence Generator

Another nifty piece of hardware and/or software which is useful for developing and testing serial applications and equipment is a character sequence generator. Such a generator produces a repeated sequence of serial line data. For example such a generator might repeat the famous “The quick brown fox ...” sentence in an an endless loop. Another common test sequence is the generation of all 8-bit codes from 0x00 to 0xFF in a loop. Such a loop contains all 7-bit ASCII and 8-bit ISO Latin 1 characters, plus the first 32 non-printable control characters and can e.g. reveal decoding errors or transmission errors. Also very common is a modem test sequence, using generic modem commands (Serial Programming:Modems and AT Commands) to build up a modem connection, send some data and tear the modem connection down in a loop.

Commercial hardware character generators provide a heap of additional features, often combined with a protocol analyzer. As such they are rather expensive. However, just like with a BOB, it is possible to build a useful DIY character sequence generator for small cash. This can either happen with software on a normal computer (some simple endless software loop sending the same data again and again to a serial interface), or with a few pieces of cheap electronic components. Some small stand-alone hardware is often more convenient in the field and in development for quick tests than e.g. a PC or laptop with some software.

A simple classic hardware character generator basically consists of a baud-rate generator, a UART (Serial Programming:8250 UART Programming), an (E)EPROM, a binary counter and a line driver (Serial Programming:MAX232 Driver Receiver). Typically, each of these components is a simple single IC. The (E)EPROM is supposed to contain the character sequence(s). The baud-rate generator drives the UART and the binary counter. The binary counter drives the address

lines of the (E)EPROM. The result is that the character sequence is produced at the data lines of the (E)EPROM. These data lines are feed into the UARTs input. The UARTs output is connected to the serial line driver. All this can be easily fitted on a small prototype board in a simple case.

A more modern hardware character generator can be build around one of these small micro controllers (e.g. Atmel AVR). This is particularly easy, since these micro controllers already contain serial interfaces, and just require a little bit of serial programming - which is the topic of this book.

# 3 Connection Types

If you wanted to do a general RS-232 connection, you could take a bunch of long wires and solder them directly to the electronic circuits of the equipment you are using, but this tends to make a big mess and often those solder connections tend to break and other problems can develop. To deal with these issues, and to make it easier to setup or take down equipment, some standard connectors have been developed that are commonly found on most equipment using the RS-232 standards.

These connectors come in two forms: A male and a female connector. The female connector has holes that allow the pins on the male end to be inserted into the connector.

## 3.1 EIA/TIA 574: DE9

This is a female DE9 (incorrectly known as “DB-9”) connector (properly known as DE9F):

The female DE-9 connector is typically used as the “plug” that goes into a typical PC. If you see one of these on the back of your computer, it is likely not to be used for serial communication, but rather for things like early VGA or CGA monitors (not SVGA) or for some special control/joystick equipment.

And this is a male “DE-9” connector (properly known as DE9M):

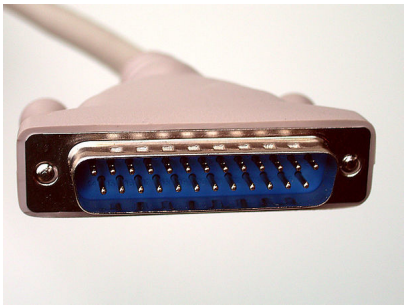
This is the connector that you are more likely to see for serial communications on a “generic” PC. Often you will see two of them side by side (for COM1 and COM2). Special equipment that you might communicate with would have either connector, or even one of the DB-25 connectors listed below.

## 3.2 RS-232C: DB-25

This is a female DB-25 connector (also known as DB25S):

This DB25S is what you normally find on an IBM compatible PC used as the parallel (printer) port. It is also on the computer end of a modem cable in older PCs that have 25 pin serial port connectors. This connector type is also used frequently for equipment that conforms to RS-232 serial data communication as well, so don't always assume if you see one of these connectors that it is always parallel. When the original RS-232 specification was written, this was the kind of connector that was intended, but because many of the pins were seldom if ever used, IBM PC compatible serial ports were later switched to the DB-9 DE9S connectors carrying all the required signals as on the DB connectors in the original IBM-PC. (Yes, this is comparatively recent equipment for this standard).

This is a male DB-25P connector (also known as DB25P):



Male DB-25 connectors are usually used on one end of a PC printer cable for parallel data communication, which is beyond the scope of this series of articles. The DB25P is also used on the the modem end of an external modem cable. You should be aware that this connector is also used for serial communications on many different types

of equipment, using many different types of communications protocols. In fact, if you have a random piece of equipment that you are trying to see how it works, you can presume that it is a piece of serial equipment. Hacking random connectors is also beyond the scope of this document, but it can be an interesting hobby by itself.

### 3.3 mini-stereo plug connector

This is a male mini-stereo plug connector:

Some digital cameras and calculators come with a cable that has a mini-stereo plug connector on the end the plugs into the camera, and a DB-9 connector on the end that plugs into the PC.

It is a poor connector, as it short circuits segments while being plugged/unplugged.

The “PicAXE” systems use

- 1: base ring: ground (pin 5 of DB9)
- 2: middle ring: serial output from PicAXE to serial input of PC (pin 2 of DB9)
- 3: tip of pin: serial output of PC to serial input of PicAXE (pin 3 of DB9)

The “LANC” systems often use a 2.5 mm stereo jack:

- base ring: ground
- middle ring: +5 to +8 VDC from the camera
- tip: normally pulled high with a resistor to +5V (idle; logical 1); pulled low (start bit and logical 0) to send commands or status. (open collector). Generally the remote control sends the first 2 byte command. The camera replies with 6 status bytes.

LANC uses 9600 bps serial.

### 3.4 RS-232D: RS232 on RJ45

RS-232D defines a standard connector much smaller than a DB-9 plug. .

(RS-232 on a RJ45 modular jack is also known as “EIA/TIA - 561”)

### 3.5 RS232 on RJ11

Is there a standard for connecting the TX, RX, GND of RS-232 to the 4 pins of a RJ11 connector ?

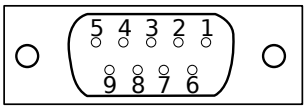
- Luhan Monat uses DB9-5 ---> RJ11-1; DB9-3 ---> RJ11-2; DB9-2 ---> RJ11-3. (RJ11-2 and RJ11-3 are the “inner pair”).
- Paul Campbell says “I wired the GND to the yellow line, TXD to the black line and RXD to the red line.”
- Russell McMahon mentions several different “standards” for wiring RS-232 to the 4 pins of a RJ11 connector or the 8 pins of a RJ45 connector.

## 4 Wiring Pins Explained

The wiring of RS-232 devices involves first identifying the actual pins that are being used.

Please note also that in the “PC COMx Port context” end of things some signals are 'inputs’ while others are 'outputs’ while in the “Modem context” those same signal names referred to now become as 'outputs’ where they were just before 'inputs’ and vice versa. That is where much confusion has arisen from over the years, as the 'Input' or 'Output' -sense- nature is not noted in most diagrams on the subject in general, yet in the real world two 'Out' pins seldom can ever work in harmony in RS-232 related +-[3-10]V stuff where the range from −3V to +3V is not a true high or low, except to possibly burden drivers towards their undesired burnout.

Here is how a **female** DB-9 connector is numbered (Note, the connector on a computer is usually a **male** connector, so it is mirrored compared to the following image):




If the numbers are hard to read, it starts at the top-right corner as “1”, and goes left until the end of the row and then starts again as pin 6 on the next row until you get to pin 9 on the bottom-left pin. “Top” is defined as the row with 5 pins.

Here are what each pin is usually defined as on the PC COMx end of things:

One thing to keep in mind when discussing these pins and their meaning, is that they are very closely tied together with modems and modem protocols.

Whenever interconnecting any serial ports it will be well to note that whatever the case, it should always follow that only one <output> should ever be tied to one or more <inputs> generally speaking. Further, be it noted that signal names at the COMx end will generally be opposite of

the <in>-<out> -sense- at the modem end of things, even though carrying the same mnemonic names.

Often you don't have a modem attached in the loop, but you still treat the equipment as if it were a modem on a theoretical level. At least such that you minimally have an <output> going to every <input> in some manner, with no two <outputs> in conflict or without any 'floating' <inputs> tied to no <output> at all.

The following are more formal explanations regarding each signal function in the general sense of its use:

## 4.1 DCD (Data Carrier Detect)

This is a signal to indicate from the communications equipment (DCE) that the phone line is still "connected" and receiving a carrier signal from the modem at the other end. Presumably well-written software or serial equipment could detect from this logic state when the telephone has been "hung up" on the other end. Null-modems often tie DCD to DTR at each end since there is no carrier signal involved.

## 4.2 RX (Receive Data)

Input to receive the data.

## 4.3 TX (Transmit Data)

The reverse of RX, this is where the terminal equipment (DTE) is transmitting serial data, using the same format and protocol that the receiver is expecting. More on the exact protocol further below. Like RX, think along the lines of "Terminal Transmit" when designing equipment that will be using this pin.

## 4.4 DTR (Data Terminal Ready)

Basically a signal from the DTE that says "Hello!, I'm ready if you are". This is a general indicator to the DCE that the terminal is ready to start sending and receiving data. If there is some initialization that needs to happen in the communications equipment, this is a way for the terminal equipment to "boot" the receiving equipment. In a null modem setup this signal is often connected to DCD, so the device signals itself that an (imaginary) carrier has been detected, indication that the transmission line is up.

## 4.5 GND (Signal Ground)

This is an interesting pin to look at. What it does is try to make a common "ground" reference between the equipment that is being connected to compare the voltages for the other signals. Normally this is a good thing, because sometimes different pieces of equipment have different power supplies and are some distance away. The not so pleasant thing about this wire is that it usually is a physical piece of copper that can conduct electricity that is not normally supposed to go down the wire, like a short-circuit or worse yet a bolt of lightning (it happens far more often that you would normally think for this sort of equipment). That can fry both the DCE as well as the DTE. Things like fiber converters and ground isolators can help prevent this from happening, but can still be something to worry about. Over short distances this is generally not a problem.

## 4.6 DSR (Data Set Ready)

This is the counterpart to DTR with the communications equipment (or computer peripheral on the serial line). When the DTR is sent as a signal, the communications equipment should change this signal to logic "1" to indicate that it is ready to communicate as well. If the DCE goes through a "boot" sequence when the DTR gets signaled, it should not signal DSR until it is complete. But many connectors "hard wire" this pin to be directly connected to the DTR pin at each end to reduce the number of wires needed in the cable. This can be useful for connecting devices using existing telephone wires, but prevents applications from using the DTR and DSR for handshaking.

## 4.7 RTS (Request To Send)

Setting the RTS signal to logic "1" indicates to the DCE that the DTE wants to send it data. Resetting the RTS signal to logic "0" indicates to the DCE that the DTE has no more data to send.

## 4.8 CTS (Clear To Send)

This is the response signal from the DCE regarding if the terminal equipment should be transmitting any data. When this signal is at logical "1", the terminal is "permitted" to transmit data. Like the DTR/DSR pins, this one can be directly connected to the RTS pin to reduce the number of wires needed, but this eliminates the possibility of hardware flow control. Some software ignores this pin and the RTS pin, so other flow control systems are also used. That will be explained when we get to actual software.

## 4.9 RI (Ring Indicator)

Again, thinking back to a telephone modem, this is a signal that indicates that the telephone is "ringing". Generally, even on a real telephone modem, this is only occasionally set to $-15$V for the signal. Basically, when you

would normally be hearing a “ring” on your telephone, this pin would be signaled. On Null-modems, often this wire isn't even connected to anything. If you really are connected to a real modem, this does have some strong uses, although there are other ways to have the terminal equipment (like a PC connected to an external modem) be informed that there are ways to communicate this information through the data pins as well. This will be covered lightly in the software section.

### 4.10 Other RS-232 Pins

There are other pins that the DB-25 has implemented that the DB-9 doesn't normally use, such as a secondary transmit and receive pin, Secondary CTS/RTS for those alternate pins, a $-15$V signal for power, a clock, and a couple of other good ideas as well. The problem with implementing all of these pins is that you also need to run separate wires, and a full set of DB-25 connectors would also mean having 25 physical wires going the full distance between the DTE and DCE. If this is more than a foot or so, it gets to be a big hassle, particularly if you are going through walls or in a more permanent setting. If the wrong wire gets clipped in the bundle, the whole thing must be restrung again, or you must go through wire testing like the old-fashioned telephone linemen used to have to do when fixing a phone distribution box. Often only three physical copper lines are used to connect the DTE to DCE, and that is simply RX, TX, and GND. The rest can be easily “faked” on the connector end in a manner sufficient for most software and hardware applications.

## 5 Baud Rates Explained

Baud and BPS (Bits Per Second) are usually not the same thing, although they are often used interchangeably, particularly in marketing literature. There are several ways to determine what the actual data rate of a particular piece of equipment is, but in popular marketing literature, or even general reference texts, they will almost always refer to “Baud Rate”, even if they are referring to bits per second.

Baud means the number of changes to the transmission media per second in a modulated signal. If each transmission event contains more than one bit of information, then Baud and BPS are not the same. E.g. if each event contains two bits (two bits modulated in an event), then the BPS of such a transmission would be twice as large as the Baud rate. This is not a theoretical case. Typical “high speed” modems use sophisticated modulation on the telephone line, where the bit rate and Baud rate differ significantly on the line. It is important to know this when you build measurement equipment, decoders (demodulators), encoders (modulators), and all sorts of transmission equipment for a particular protocol.

However, software developers typically like to ignore the difference of bit rate and baud rate, because a bit can either have the value true or false - an “event” (a bit) always only has two possible states. They have no basic unit which can e.g. hold four different states. In other words, on the software site the modulation has already been flattened by the demodulator. If a modulation was used which can e.g. transmit 8 bits in an event, the software developer sees them already as a series of 8 consecutive bits, each either true or false. The demodulator took care of that. When it got an event it turned the single 8-bit event into eight single-bit events. Software developers don't see the original single entity with 256 different states (voltages, phases). Since the modulation has been flattened they don't experience the difference between Baud rate and bit rate any more. This is not the fault of the people who defined a Baud or a BPS. It is just a (welcome) limitation of digital computer hardware.

Baud is actually a shortened term named in honor of Émile Baudot, a French inventor of early teleprinter machines that replaced the telegraph key using Morse Code. Basically two typewriters that could be connected to each other with some wires. He came up with some of the first digital character encoding schemes, and the character codes were transmitted with a serial data connection. Keep in mind this was being done largely before computers were invented. Indeed, some of these early teleprinter devices were connected to the very first computers like the ENIAC or UNIVAC, simply because they were relatively cheap and mass produced at that point.

In order for serial data communication to happen, you need to agree on a clock signal, or baud rate, in order to get everything to be both transmitted and received properly. This is where the language purists get into it, because it is this clock signal that actually drives the “baud rate”. Let’s start more at the beginning with Émile Baudot’s teleprinters to explain baud rate.

Émile’s early teleprinters used 5 data bits and 1 stop bit to transmit a character. We will go onto formatting issues in a second, but what is important is that six signals are sent through a wire in some fashion that would indicate that a character is transmitted. Typically the equipment was designed to run at 50 baud, or in other words the equipment would transmit or receive a “bit” of data 50 times per second. Not coincidentally, French power systems also ran on an alternating current system of 50 Hz, so this was an easy thing to grab to determine when a new character should be transmitted.

Teleprinters evolved, and eventually you have Western Union sending teleprinter “cablegrams” all around the world. If you hear of a TELEX number, this is the relic of this system, which is still in use at the present time, even with the Internet. By rapidly glossing over a whole bunch of interesting history, you end up with the United States Department of Justice (DOJ) in a lawsuit with AT&T. Mind you this was an earlier anti-trust lawsuit prior to the

famous/infamous 1982 settlement. The reason this is important is because the DOJ insisted that Western Union got all of the digital business (cable grams... and unfortunately this got to be read as computer equipment as well), and AT&T got modulated frequencies, or in other words, you could talk to your mother on Mother's Day on their equipment. When computers were being built in the 1950s, people wanted some way to connect different pieces of computer equipment together to "talk" to each other. This finally resulted in the RS-232 standard that we are discussing on this page.

While Western Union was permitted to carry digital traffic, often the connections weren't in or near computer centers. At this time AT&T found a loophole in the anti-trust settlement that could help get them into the business of being a "carrier" of computer data. They were also offering to transmit computer data at rates considerably cheaper than Western Union was going to charge. Hence, the modem was born.

## 5.1 Modems Explained

The long description of a modem is a "Modulator/Demodulator", and this description is important. Since AT&T could only carry "tones", like music from a radio network or the voice of your mother, they created a device that would electronically create "music" or "tones" that could be carried on their network. They would then take a computer "1" or "0" and "modulate" the bit to a frequency, like say 2600 Hz. (The exact tones varied based on baud rate and other factors, but there were exact frequency specs here.) A matching device would be able to look for that "note" or "tone" in the "music" and be able to convert that back to a computer "1" or "0", or in other words, demodulate the music. Since all you and your buddy on each end of the telephone are only playing music to each other, it was legal for AT&T to have that music on their network. That only computers could possibly understand this music is besides the point, and the DOJ turned a blind eye on the whole practice, despite objections from Western Union.

The original modems you could rent were AT&T Bell 103 modems. These were clunky boxes about the size of a shoe box that had a bunch of switches on the outside and an RS-232 cable that connected to the computer equipment you were using. These boxes were designed for the old-fashioned handset telephones and had pieces of rubber that would go around the "speaker" and "mic" portion of the telephone (no direct copper connection to the telephone equipment back then). If you wanted to dial the telephone, you had to use the rotary dial on the phone itself... the computer didn't have access to that sort of equipment. Keep in mind that the FCC regulated just about everything that happened with phone equipment, and AT&T owned everything related to telephones. You even had to "rent" the modem from AT&T, and that rental charge was on your monthly phone bill.

The Bell 103 was originally 110 baud, although it eventually had a switch to "move up" to 220 baud. 300 baud modems were also fairly common throughout the 1960's and 1970's. Keep in mind that AT&T (or your local phone company) was the only company you could even rent a modem from, whether you wanted one or not. By 1982, modems were so commonly used and the POTS telephone network so widespread that this same system of sending "music" over the telephone has been preserved, even though the legal reasons for doing it are no longer valid. With the advent of ISDN and DSL lines, this is no longer the case and the phone companies are now sending pure digital signals instead. This is also why DSL lines can carry much more data than an ordinary phone line, even though it is the same pair of copper wires going into your home.

When modems started going to very high speeds, they hit a brick wall of sorts. It was decided back in the 1950's that telephone equipment would only have to carry tone signals going to about 10kHz. For normal voice conversations this is sufficient, and you can even tell the difference between a man and a woman on the telephone. The problem comes in that this means the highest normal "baud rate" that you can send over a home telephone network is about 9600 baud, usually about 4800 baud, because the telephone equipment itself is going to be dropping "bits" as you switch from one tone to another. Without going into the heavy math, you need to have at least one full "sound wave" in order to be able to distinguish one tone or note from another. Modem manufacturers did think of something else that could be done to overcome this limitation, however. Instead of just sending one tone at a time, you could play a whole "chord", or several distinct tones at the same time. Finally back to baud vs. bits per second. With higher speeds, instead of simply sending only one bit, you are sending two or as many as sixteen bits at the same time with varying "chords" of "music". This is how you get a 56K BPS modem, even though it is still only transmitting at 9600 baud.

More about modems in Serial Programming:Modems and AT Commands.

# 6 Signal Bits

There are four sets of transmission bits that are used in the RS-232 standard. The positioning of these bits in the RS-232 data stream is all that distinguishes one bit from the other. This is also where serial communication really hits the "metal", because each bit follows in a sequence, or in a serial fashion. All of the other wires, pins, baud rate, and everything else is to make sure that these bits can be understood. Keep in mind that at this point the entire protocol is based on the transmission of a single character. Multiple characters can be sent, but they are a sequence of single character transmission events. How the characters relate is based on what the software does

with the data on the next protocol “layer”.

## 6.1 Start Bit

When a transmission line is not sending anything, it remains in a logical state of “1”, or −15V on the wire. When you want to send a character, you start by changing the voltage to +15V, indicating a logical “0” state. Each subsequent bit is based on the baud rate that is established for communication between each device. This bit signals that the receiving device should start scanning for subsequent bits to form the character.

## 6.2 Data Bits

This is the primary purpose of serial communications, where the data actually gets sent. The number of bits here can vary quite a bit, although in current practice the number of bits typically transmitted is eight bits. Originally this was five bits, which was all that the early teleprinters really used to make the letters of the Alphabet and a few special characters. This has implications for Internet protocols as well, because early e-mail systems transmitted with only seven bits when they were connected over some RS-232 links. This worked because the early character encoding schemes, mainly ASCII, only used seven bits to encode all characters commonly used for the English language. Because computer components work best on powers of 2 (2,4,8,16,32, etc.), eight bits became more commonly used for data storage of individual characters. Unicode and other coding schemes have moved this concept forward for languages other than English, but eight bits still is a very common unit for transmitting data, and the most common setting for RS-232 devices today.

The least significant bit (LSB) is transmitted first in this sequence of bits to form a character.

## 6.3 Parity Bit

To help perform a limited error check on the characters being transmitted, the parity bit has been introduced. Parity can detect some transmission errors but not correct. The value of the parity bit depends on the number of bits set to “1” in the string of data bits.

There are four different kinds of parity configuration to consider:

### 6.3.1 Odd Parity

When the sum of bits ends up coming up with an odd number (like the sequence 01110101), this bit will be set to a logical state of “1”.

### 6.3.2 Even Parity

This uses the formula of trying to determine if there are an even number of bits set to “1”. In this regard, it is the exact opposite state of the Odd Parity. For e.g., for a frame with seven bits that has an odd number of ones, the parity bit will be set to one. So essentially, the entire byte, including parity must have an even number of ones for even parity.

### 6.3.3 Mark Parity

Using this concept, the transmission protocol is essentially ignoring the parity bit entirely. Instead, the transmission configuration is sending a logical “1” at the point that a parity bit should be sent, regardless of if the sequence should have an odd or even count. This configuration mode is useful for equipment that may want to be testing parity checking software or firmware in the receiving equipment.

### 6.3.4 Space Parity

The opposite of Mark parity, this sends a logical “0” for the parity checksum. Again, very useful for equipment diagnostics.

### 6.3.5 Parity None

This isn't really a parity formula, but rather an acknowledgment that parity really doesn't work, so the equipment doesn't even check for it. This means the parity bit isn't even used. This can cause, in some circumstances, a slight increase in the total data throughput. More on that below.

## 6.4 Stop Bits

This really isn't a bit at all, but an agreement that once the character is sent that the transmitting equipment will return to a logical “1” state. The RS-232 specification requires this logical state of “1” to remain for at least one whole clock cycle, indicating that the character transmission is complete. Sometimes the protocol will specify two stop bits. One reason that this might be done is because the clock frequencies being used by the equipment might have slightly different timing, and over the course of hundreds or thousands of characters being transmitted the difference between two clocks on the two different pieces of equipment will cause the expected bits to be shifted slightly, causing errors. By having two stop bits the transmission is slightly slower, but the clock signals between the two pieces of equipment can be coordinated better. Equipment expecting one stop bit can accept data transmitted by equipment sending two stop bits. It won't work the other way around, however. This is something to try if you are having problems trying to get two pieces

of equipment to communicate at a given baud rate, to add the second stop bit to the transmitter.

### 6.5 Data Transmission Rates

We got into a discussion of baud rate vs. bits per second. Here is where baud as the number of bits being transmitted is still off, even if the nominal bits per second is also the same as the baud rate. By adding start bits, stop bits, and parity bits, that is going to add overhead to the transmission protocol. All digital transmission protocols have some sort of overhead on them, so this shouldn't be that much of a surprise. As we get more into data packets and other issues, the actual amount of data being transmitted will drop even further.

Keep in mind that if you are transmitting with 6 data bits, 2 Stop bits, and Even Parity, you are transmitting only six bits of data and four other bits of extra information. That means even with 9600 baud, you are only transmitting 5,760 bits of data per second. This really is a big difference, and that is still only raw bits once it gets through the actual serial communications channel. A more typical 8 data bits, 1 Stop Bit, No Parity will be a little bit better at 9600 baud, with eight bits of data and only two bits used for overhead. That gives a total throughput of 7,680 bits per second. A little bit better, but you can't simply presume that the baud rate indicates how much data is going to be transmitted.

## 7 Relationship of Baud Rate to Maximum Distance

There are physical limits to how far serial data communication can occur over a piece of wire. When you apply a voltage onto a wire it takes time for that voltage to traverse the wire, and there are other unstable conditions that happen when you send a “pulse” down the wire and change voltages too quickly. This problem is worse as wires become longer and the frequency (i.e. baud rate) increases. This distance can vary based on a number of factors, including the thickness of the wires involved, RF interference on the wires, quality of the wires during the manufacturing process, how well they were installed... e.g., are there any “kinks” in the wires that force it into a sharp bend, and finally the baud rate that you are transmitting the data.

This table presumes a fairly straight and uniform cable that is typical for most low-voltage applications (i.e., not a power circuit that uses 110V to run your refrigerator, toaster, and television). Typically something like a CAT-5 cable (also used for local networks or phone lines) should be more than sufficient for this purpose.

The distance limitation can be mitigated. There are “short haul modems” that can extend this distance to several miles of cable. There are also telephone lines, or conventional modems, and other long-distance communications techniques. There are other ways to handle data in situations like this, and those signals can be converted to simple RS-232 data formats that a typical home computer can interpret. Distance still can be a limiting factor for communication, although when you are talking about distances like to Saturn for the Cassini mission, serial data communication has other issues involved than just data loss due to cable length. And yes, NASA/ESA is using serial data communication for transmitting those stunning images back to Earth.

## 8 External References

- RS-232 wiring standards explained
- RS-232 connection types explained
- Wikipedia article on RS-232
- RS-232 standards explained by HW-Server
- Serial Pinouts (D25 and D9 Connectors) (also has more technical information about the UARTs used in PCs)
- RS232 Connections, and wiring up serial device has several diagrams, including one showing how to let one PC monitor the serial communication between 2 other RS232 devices.
- Lammert Bies, RS232 Specifications and standard Includes technical specs on RS-232 signals and more detailed information about parity checking.
- Tronisoft's Printable ASCII Serial Port Crib Sheets
- jSSC library (Java Simple Serial Connector). Work under Win32 and Win64

## 9 Other Serial Programming Articles

Typical RS232-Hardware Configuration

**Serial Programming**: Introduction and OSI Network Model -- RS-232 Wiring and Connections -- Typical RS232 Hardware Configuration -- 8250 UART -- DOS -- MAX232 Driver/Receiver Family -- TAPI Communications In Windows -- Linux and Unix -- Java -- Hayes-compatible Modems and AT Commands -- Universal Serial Bus (USB) -- Forming Data Packets -- Error Correction Methods -- Two Way Communication -- Packet Recovery Methods -- Serial Data Networks -- Practical Application Development -- IP Over Serial Connections

# 10 Text and image sources, contributors, and licenses

## 10.1 Text

- **Serial Programming/RS-232 Connections** *Source:* https://en.wikibooks.org/wiki/Serial_Programming/RS-232_Connections?oldid=3010842 *Contributors:* DavidCary, Robert Horning, Panic2k4, Guanabot~enwikibooks, Darklama, Renffeh, Jomegat, Jguk, Yuriybrisk, Webaware, Wajidstar, Dallas1278, QuiteUnusual, Adrignola, Sandcat01, Theodore.cackowski, Stryn, BeakerGordon and Anonymous: 90

## 10.2 Images

- **File:9_pin_d-sub_connector_male_closeup.jpg** *Source:* https://upload.wikimedia.org/wikipedia/commons/7/78/9_pin_d-sub_connector_male_closeup.jpg *License:* Public domain *Contributors:* Photo taken by Mike1024. *Original artist:* User Mike1024
- **File:KL_Break_Out_Box_RS-232.jpg** *Source:* https://upload.wikimedia.org/wikipedia/commons/f/fe/KL_Break_Out_Box_RS-232.jpg *License:* GFDL *Contributors:* Camera: Canon EOS 400D *Original artist:* Konstantin Lanzet
- **File:Klinkenstecker_stereo_3.5mm.jpg** *Source:* https://upload.wikimedia.org/wikipedia/commons/c/c1/Klinkenstecker_stereo_3.5mm.jpg *License:* CC BY-SA 2.5 *Contributors:* Own work *Original artist:* Afrank99
- **File:Parallelport.jpg** *Source:* https://upload.wikimedia.org/wikipedia/commons/2/24/Parallelport.jpg *License:* CC-BY-SA-3.0 *Contributors:* Own work *Original artist:* Afrank99
- **File:RS-232.jpeg** *Source:* https://upload.wikimedia.org/wikipedia/commons/8/87/RS-232.jpeg *License:* CC-BY-SA-3.0 *Contributors:* ? *Original artist:* ?
- **File:RS232_Buchse_9pol_female.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/c/c3/RS232_Buchse_9pol_female.svg *License:* Public domain *Contributors:*
- RS232_Buchse_9pol_female.png *Original artist:* RS232_Buchse_9pol_female.png: StefB at de.wikipedia.
- **File:Scsi_extern_db25_st.jpg** *Source:* https://upload.wikimedia.org/wikipedia/commons/8/8d/Scsi_extern_db25_st.jpg *License:* CC BY-SA 2.0 de *Contributors:* Own work *Original artist:* User Smial on de.wikipedia
- **File:Wikipedia-logo.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/6/63/Wikipedia-logo.png *License:* GFDL *Contributors:* based on the first version of the Wikipedia logo, by Nohat. *Original artist:* version 1 by Nohat (concept by Paullusmagnus);

## 10.3 Content license

- Creative Commons Attribution-Share Alike 3.0