Keysight Technologies
RS-232 Troubleshooting

Application Note

KEYSIGHT TECHNOLOGIES

Introduction

In the course of dealing with personal computers, you may use the RS-232 serial interface. This application note will describe RS-232 at a basic level, with an orientation towards Windows-based instrument programming.

1. An Overview of RS-232

The first question that needs to be addressed is: what precisely is a "serial" interface?

Consider a computer connected to an instrument or other "remote" device. One of the simplest possible communications schemes is shown in Figure 1.

Each device sends data bits coded as electrical pulses, with a "0" corresponding to a low voltage and a "1" a high voltage, to the other over a dedicated line, using a shared ground line. Using separate lines to transmit and receive data allows both devices to send data simultaneously without interference, at least in principle.

For example, to send a byte to the remote device, the computer would have to send as shown in Figure 2.

There are a wide variety of serial communications schemes. The most popular is RS-232, which is in fact so universal that it is often simply referred to as "serial." RS-232 defines various mechanical and electrical specs for serial communications.

RS-232 defines legal voltage levels as follows:

	Output	Input
0 (mark)	+5 to +15 volts DC	+3 to +15 volts DC
1 (space)	–5 to –15 volts DC	–3 to –15 volts DC

The terms "mark" and "space" are ancient nomenclature for a "0" and "1" that are still in occasional use.

RS-232 also defines a transmission format for sending data over a serial link. Suppose you want to send a byte (or, more generally, a "word") of data over a serial connection from your computer to a remote device.

Now further suppose that several bits—or all of them, for that matter, have the same value. How can the remote device tell which bit is which? Where does one bit start and the other end?

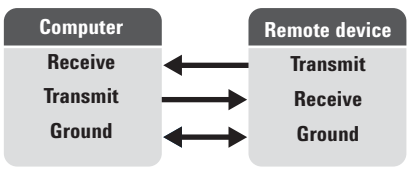

Figure 1

Figure 2

The only way possible way under this scheme is for both the computer and the remote device to agree on how long each bit remains on the line. For example, let's say each bit stays on the line for 1/9600 of a second. Then the remote device can count from the middle of each bit time to the middle of the next bit time and be reasonably sure that it had obtained a valid value of each bit.

Of course, the timing on the computer may not be perfectly matched to the timing on the remote device, but since there's only 8 bits being sent at a time, there won't be time to get out of step.

In serial communications, the inverse bit time is called the "baud rate". Baud rates can be any value in principle, but in practice and by custom the baud rate is usually set to certain values: 19,200, 9600, 4800, 2400, 1200, 600, 300, or 110 baud.

Baud rate is sometimes thought of as being the same as the bit transmission rate of the serial link, but that's not precisely true, since there is no guarantee that words will be sent in a continuous stream.

Furthermore, in practice there's some additional overhead. Just sending 8 bits in this fashion works fine, until you ask the question: what happens if the first bits are 0, not 1? How can the remote figure out where the byte starts?

The answer is that it can't, so to prevent this problem, an extra bit that is always set to 1 is tacked on in front of the other serial bits. This initial bit is called a "start bit."

The data bits are also, by convention, followed by "stop bits" that are set to 1, and indicate the end of the word. There can be 1, 1.5, or 2 stop bits. The computer and the remote device have to agree on how many stop bits are sent. Using more stop bits gives a device a little more time to process words as they are sent in. The number of stop bits can vary, but the number of start bits is always 1.

So under RS-232 words of data are sent with 1 start bit, and 1 or more stop bits. The data words don't have to be 8 bits; they can be 5, 6, 7, or 8 bits, though these days really only 7 or 8 bits are used.

Words can also be sent with an optional "parity" bit. This is an extra bit that can be tacked on behind the data bits as an error check—either to make the total number of "1" bits even or to make the number of "1" bits odd.

Parity is not a very useful form of error checking, and while it can be used in nearly all serial communications systems, it usually isn't. You can also specify that the parity bit always be set to 1 or 0.

Okay, that nails down the fundamental RS-232 parameters—baud rate, stop bits, word size, and parity. You'll need to specify these parameters to configure your serial communications.

Now that we understand how to speak, the next step is to consider what to say—that is, how conversations are conducted over serial.

There really aren't any fixed rules, but some general ideas can be presented. Let's go back to our simple serial system and consider how such a conversation might take place. See Figure 1.

The simplest conversation protocol possible is for the remote device to act as a slave to the computer*: the computer sends a command; the remote device makes a response. The advantage of this is that the computer is completely in control of the communications. The remote device cannot send anything when the computer is not ready for it, and communications will not be confused.

The subtle question here is: how does the remote device know when it has received a command and not merely part of one?

Remote commands could be defined either as binary codes or as ASCII strings, but binary codes tend to be inconvenient, so ASCII strings are more common. These ASCII-based commands ideally should have natural-language syntax—like "MEASURE," "STATUS," and so on.

Since these commands may have various options or defaults, their length may be ambiguous, so to allow the remote device to determine where the command ends, "end-of-line terminator," or simply "terminator," characters are tacked on to the end.

The most common terminator is line-feed (LF—ASCII code 10), or carriage-return line-feed (CR-LF—ASCII code 13 and 10)—though some devices use just a CR or even a NULL (ASCII 0) character:

MEASURE:VOLTS<LF>

* Note that a serial connection can be "half-duplex"—meaning that the computer can talk to the remote, or the remote can talk to the controller, but they can't both talk to each other at the same time—as opposed to "full-duplex," in which they both can talk at the same time.

The remote device could send back a response either in a fixed-length binary format (which is fast, but hard to read and interpret)—or encoded as ASCII. If more than one data item is returned as ASCII, the items could be separated with commas, and the full string terminated with a LF:

32,45,1,128,512,64<LF>

Once you understand how a conversation is performed over RS-232, the next problem is to make sure it is reliable—that is, that one side is listening attentively while the other is talking, and that no information is lost. It is easy to lose data in RS-232 communications because, as defined so far, one side can talk away and never realize that everything it is sending is being lost because the other side can't keep up, or is otherwise distracted.

For example, suppose the computer sends a command to the remote device, but the remote device responds so fast the computer isn't ready to read the response. If the RS-232 implementation on the computer has "buffering"—that is, it can store up a block of data even when the computer isn't ready to read it—this isn't such a problem, but a buffer can overflow if the computer never attends to it, and not all computer RS-232 implementations have buffering.

In the absence of buffering, the simplest way to avoid an "input overflow" is for the remote device to wait a short period of time before responding. Some remote devices have a DIP switch or jumper settings to allow you to specify a particular delay time. Similarly, if the remote device responds with multiple lines of response data, you may be able to select a similar delay between each line to give the computer a chance to receive them all.

A related scheme to allow reliable transfer of multiple lines of ASCII is known as "prompting." Every time the computer receives a line of ASCII from the remote device, the computer sends a "prompt" string (say, a CR) back to the remote device to tell it to send another line.

More generally, the computer should have some means of telling the remote device to be quiet for a while until the computer has received the data and is ready for more.

In RS-232, this capability is known as "handshaking" or "flow control." Given a three-wire serial system as we have defined it so far, there is a scheme known as "XON-XOFF" flow control that is often used in RS-232 communications.

In this scheme, the remote device sends data until the computer starts to get too full. The computer then sends a character to tell the remote to be quiet—an XOFF ("transmit off") character, usually defined as DC3 (ASCII code 19), and the remote device stops sending. When the computer wants more data, it sends a character to tell the remote device to start sending again—an XON ("transmit on") character, usually defined as DC1 (ASCII code 17)—and the remote device starts sending again.

The problem with XON-XOFF flow control is that its resolution is "grainy." It can't be used to control the data flow on a word-by-word basis, it can only control data flow in terms of blocks of data, and generally implies some level of buffering (as well as full-duplex communications).

There is an alternative. As defined so far, our serial link only uses three wires: transmit, receive and ground. However, RS-232 defines a large number of "control lines" beyond those three lines that can be used for flow control.

These control lines were originally defined for interfacing to an external modem, which is of no concern in this document, and so a detailed discussion of the actual meanings of these lines is not particularly useful. They can simply be seen as a set of output control lines and input status lines.

See Figure 3 for the PC's 9-pin RS-232 pinout.

These control lines can be used to implement flow control schemes. For example, the RTS line on the computer could be wired to the CTS on the remote device (and the reverse). When the computer wants to receive data, it sets RTS, and when it wants to stop receiving data, it clears RTS; the remote device checks the status of its CTS input on a word-by-word basis to see if it should send or not.

This is known as "RTS-CTS flow control." The DTR and DSR lines are also used for the same purpose. The other control lines may be used as auxiliary controls.

The 9-pin connector used on a PC is only one of a number of connector formats. There is also a 25-pin format, and in principle the connector could be of either gender—the PC connector is a male—with a wide variety of wiring schemes.

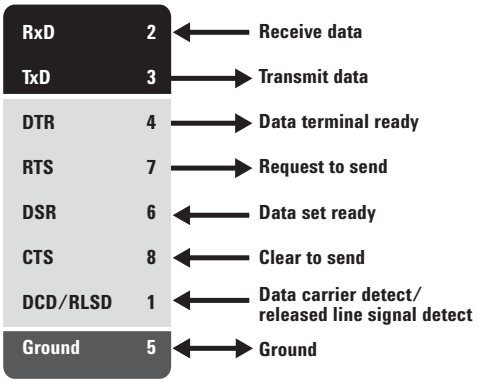

Figure 3

A further confusing factor is that a connector may be a DTE ("data terminal equipment") or DCE ("data communications equipment") connection, a holdover from RS-232's definitions for use with modems. On a DTE device, connections mean what they seem to mean: "transmit data" is an output, while "receive data" is an input—the PC connector is a DTE. On a DCE device, *all the meanings of the connections are reversed!*—"transmit data" is an *input* and "receive data" is an *output*.

The variation in connector and cable wiring was a particular problem in the past, and made figuring out what cable to use extremely difficult, leading to a description of RS-232 as the "bunch of wires" interface. However, the predominance of the PC has made its 9-pin format something of a standard, and most modern RS-232 equipment is easy to cable up. Trying to figure out the cabling can be a nasty problem with older equipment, however.

2. RS-232—Real-World Issues

You should now have a grasp of the basic concepts of RS-232 operation:

- Baud rate, word size, start bits, stop bits, and parity.
- Command and data formats.
- Half- and full- duplex, buffering, flow control, control lines, DCE and DTE.
- Connection schemes.

Given this knowledge, the ideal RS-232 instrument should have the following characteristics:

- The ability to select from a reasonable set of baud rates, word sizes, stop bits, and parity options via a DIP switch, jumper, or front-panel options.
- English-like commands in ASCII format, using a CR-LF or LF terminator.
- ASCII data formats using comma separators, using a CR-LF or LF terminator.
- Buffering.
- The capability to select turnaround delays, or XON-OFF, RTS-CTS, DTR-DSR, or no flow control via a DIP switch, jumper, or front-panel options.
- A PC-compatible pinout for predictability.

RS-232 has become easier to deal with in recent years, due to the influence of the personal computer. Most devices will use a PC-compatible connection and will default to 9600 baud, 1 stop, no parity.

This makes life much simpler, but the other items remain unpredictable, and for older RS-232 instruments all bets are off. The problem is that a serial interface is very cheap and easy to implement. The result is that a serial instrument can operate in any way the designers like. A serial instrument may have:

- A fixed baud rate.
- A binary command set and data formats.
- Any sort of terminator character.
- No, or very limited, provisions for flow control.
- A 9- or 25-pin connector in either gender, using a virtually arbitrary wiring scheme, and defined as DTE or DCE (though as mentioned this isn't such a problem any more).
- Peculiar dependencies on the logic state of RS-232 control pins you may not have control over in your programming language.

Confronted with such difficulties, you may need to be resourceful to get the remote device to work. You can compare it to opening a combination lock: if you don't know all the right ways to turn the knob, you won't get in.

A few final comments:

- A PC normally has two 9-pin RS-232 ports, designated as COM1 and COM2. Under normal circumstances, two more RS-232 ports can be added into a PC as COM3 and COM4.

Serial "multiplexer" cards are also available that offer a large number of serial ports—with the catch that the computer can only use one at a time. In reality, this isn't much of a catch, since a single-CPU machine can only *do* one thing at a time, and

some of the serial multiplexer cards have buffer RAM that allows them to accumulate inputs while the CPU is off doing other things, so nothing is lost.

- If you wish to link two PCs together over serial you will need what is called a "null modem" cable —basically a DTE-DTE connection with wiring that reverses the connections.

- When you are playing with RS-232, you will sometimes hear about a BREAK. Executing a BREAK puts the RS-232 line in a "space" (zero) mode for "longer than a single RS-232 data frame," possibly something like ten serial frames. It clears the line so the remote device can sync up again.

- Some RS-232 instruments assume they are connected to a computer terminal, and have protocols that are very difficult to handle. For example, they may send a response string, followed by a terminator, followed by a prompt (like "≥"), or even send complete display screens that assume a particular type of computer terminal, such as DEC VT100.

Even worse is an RS-232 instrument that performs "remote echo"—that is, every time the instrument gets a character, it echoes it back.

- Most of the low-level RS-232 protocols—setting up start and stop bits, handling flow control, and so on—are handled by a chip known as a "UART," for "universal asynchronous receive transmit." You'll see this term mentioned occasionally in serial documentation.

- There are variations on RS-232, such as RS-423, RS-485, and particularly RS-422. There is also an antique scheme known as "current loop" that dates from the era of teletype terminals.

From the user's point of view they are similar to RS-232, except they use different output devices. In some cases they allow longer and (in principle) faster connections. RS-485 also allows communication with multiple devices on the same bidirectional connection.

- There are lots of higher-level communications protocols that can be used on serial—Kermit, XMODEM, UMODEM, and others—that provide for data integrity and reliable communications; discussion of these protocols is beyond the scope of this document, but they are mentioned here for the sake of completeness.

- If you are trying to interface an RS-232 instrument to a PC, the best thing to start with is Keysight Technologies, Inc. Connection Expert and its Interactive IO utility to see if you can establish communications at all.

The first thing you need to do after that is ensure that your cable is actually the right one. Vendors can often recommend a cable, but in the worst case you may have to actually do some wiring on your own.

Then you can start tinkering with communications parameters to see what you can get to work. Note that you should turn off all handshaking at first. You'll probably get errors, but at least you can determine if you are talking to the device.

- People who spend time working with a variety of RS-232 devices usually acquire a set of tools to make the task easier. Any RS-232 troubleshooter will usually have a set of "sex-changers" (or "gender-benders") to allow connection of two male or two female connectors, and 9-to-25-pin connectors.

- Some devices that have multiple interfaces have to be configured to communicate over RS-232. For instance, the Keysight 34401 DMM can be set from the front panel to work as RS-232 or GPIB; if you have it set to GPIB, it doesn't work very well with RS-232. (The 34401 is kind enough to announce on its display on power up whether it is set to RS-232 or GPIB.)

Please do not underestimate RS-232 programming problems. For some reason the topic seems to give newcomers to the issue a false impression of simplicity. It can be quite simple in some cases—when the remote device is well-behaved and well-documented and you are using reasonable controlling software—but if you are performing serial interfacing, you best be prepared for a struggle.

3. Troubleshooting RS-232 Problems

If you're having trouble getting an RS-232 connection to an instrument to work, you will need to work through methodical troubleshooting steps:

- First, make sure you have the right RS-232 cable with the right wiring connected to your device.
- Second, make sure that nothing else , like a printer, mouse, or other applications program, is using the RS-232 port. Try selecting the port with a terminal emulator if you are having troubles.
- Third, ensure that both the instrument and your program have the same serial settings—baud rate, word size, stop bits, parity, handshaking, and so on. Conventional settings are 9600 baud, 8 bits, 1 stop, no parity Handshaking mode varies, but buffering is common these days, so "no handshaking" is a good place to start.
- Fourth, make sure that you understand the command set of the device and its data formats. You will not in general be able to communicate with an RS-232 device by guesswork. Unfortunately, some RS-232 manuals are extremely cryptic and obscure.

If the device is compatible with the 488.2 common-command subset (this is often the case if the device has both RS-232 and GPIB interfaces), then you can assume that it does support a small set of standard commands. Try sending a "*RST;*CLS" to see if it clears the device, and try to query it for its ID string with an "*IDN?" query.

If this doesn't get you anywhere, some troubleshooting steps are in order.

Try using a terminal emulator to see if you can send simple commands or otherwise communicate with the instrument. This is a very useful and highly recommended step if you are having problems.

If it seems that you can communicate between the terminal emulator and the instrument then it is likely that there is some misunderstanding of command and data formats. If the manuals seem ambiguous on the command and data formats, then you may have to do some probing.

If you don't seem to be able to read back data, you might try reading back data one byte at a time, and display both its ASCII code value and the corresponding character.

myKeysight

myKeysight

www.keysight.com/find/mykeysight
A personalized view into the information most relevant to you.

www.axiestandard.org
AdvancedTCA® Extensions for Instrumentation and Test (AXIe) is an open standard that extends the AdvancedTCA for general purpose and semiconductor test. Keysight is a founding member of the AXIe consortium. ATCA®, AdvancedTCA®, and the ATCA logo are registered US trademarks of the PCI Industrial Computer Manufacturers Group.

www.lxistandard.org
LAN eXtensions for Instruments puts the power of Ethernet and the Web inside your test systems. Keysight is a founding member of the LXI consortium.

www.pxisa.org
PCI eXtensions for Instrumentation (PXI) modular instrumentation delivers a rugged, PC-based high-performance measurement and automation system.

Three-Year Warranty
www.keysight.com/find/ThreeYearWarranty
Keysight's commitment to superior product quality and lower total cost of ownership. The only test and measurement company with three-year warranty standard on all instruments, worldwide.

Keysight Assurance Plans
www.keysight.com/find/AssurancePlans
Up to five years of protection and no budgetary surprises to ensure your instruments are operating to specification so you can rely on accurate measurements.

www.keysight.com/quality
Keysight Technologies, Inc.
DEKRA Certified ISO 9001:2008
Quality Management System

Keysight Channel Partners
www.keysight.com/find/channelpartners
Get the best of both worlds: Keysight's measurement expertise and product breadth, combined with channel partner convenience.

For more information on Keysight Technologies' products, applications or services, please contact your local Keysight office. The complete list is available at:
www.keysight.com/find/contactus

Americas

Canada	(877) 894 4414
Brazil	55 11 3351 7010
Mexico	001 800 254 2440
United States	(800) 829 4444

Asia Pacific

Australia	1 800 629 485
China	800 810 0189
Hong Kong	800 938 693
India	1 800 112 929
Japan	0120 (421) 345
Korea	080 769 0800
Malaysia	1 800 888 848
Singapore	1 800 375 8100
Taiwan	0800 047 866
Other AP Countries	(65) 6375 8100

Europe & Middle East

Austria	0800 001122
Belgium	0800 58580
Finland	0800 523252
France	0805 980333
Germany	0800 6270999
Ireland	1800 832700
Israel	1 809 343051
Italy	800 599100
Luxembourg	+32 800 58580
Netherlands	0800 0233200
Russia	8800 5009286
Spain	0800 000154
Sweden	0200 882255
Switzerland	0800 805353 Opt. 1 (DE) Opt. 2 (FR) Opt. 3 (IT)
United Kingdom	0800 0260637

For other unlisted countries:
www.keysight.com/find/contactus
(BP-07-10-14)

KEYSIGHT TECHNOLOGIES

Published in USA, July 31, 2014
5989-6580EN
www.keysight.com