

## Serial Communication General Concepts

### Overview

This DevZone is serial communication primer that will explain the fundamentals of serial communication, along with the most common protocols.

### Table of Contents

1. [What is Serial](#)
2. [What is RS-232?](#)
3. [What is RS-422?](#)
4. [What Is RS-485?](#)
5. [What is Handshaking?](#)
6. [Related Links](#)

### What is Serial

The concept of serial communication is simple. The serial port sends and receives bytes of information one bit at a time. This is slower than parallel communication, which allows the transmission of an entire byte at once, it is simpler and can be used over longer distances. For example, the IEEE 488 specifications for parallel communication state that the cabling between equipment can be no more than 20 meters total, with no more than 2 meters between any two devices; serial, however, can extend as much as 1200 meters.

Typically, serial is used to transmit ASCII data. Communication is completed using 3 transmission lines: (1) Ground, (2) Transmit, and (3) Receive. Since serial is asynchronous, the port is able to transmit data on one line while receiving data on another. Other lines are available for handshaking, but are not required. The important serial characteristics are baud rate, data bits, stop bits, and parity. For two ports to communicate, these parameters must match:

1. **Baud rate** is a speed measurement for communication. It indicates the number of bit transfers per second. For example, 300 baud is 300 bits per second. When we refer to a clock cycle, in the context of serial, we mean the baud rate. For example, if the protocol calls for a 4800 baud rate, then the clock is running at 4800Hz. This means that the serial port is sampling the data line at 4800Hz. Common baud rates for telephone lines are 14400, 28800, and 33600. Baud rates greater than these are possible, but these rates reduce the distance by which devices can be separated. These high baud rates are used for device communication where the devices are located near one another.
2. **Data bits** are a measurement of the actual data bits in a transmission. When the computer sends a packet of information, the amount of actual data may not be a full 8 bits. Standard values for the data packets are 5, 7, and 8 bits. Which setting you choose depends on what information you are transferring. For example, standard ASCII has values from 0 to 127 (7 bits). Extended ASCII uses 0 to 255 (8 bits). If the data being transferred is simple text (standard ASCII), then sending 7 bits of data per packet is sufficient for communication. A packet refers to a single byte transfer, including start/stop bits, data bits, and parity. Since the number of actual bits depend on the protocol selected, the term packet is used to cover all instances.
3. **Stop bits** are used to signal the end of communication for a single packet. Typical values are 1, 1.5, and 2 bits. Since the data is clocked across the lines and each device has its own clock, it is possible for the two devices to become slightly out of sync. Therefore, the stop bits not only indicate the end of transmission but also give the computers some room for error in the clock speeds. The more bits that are used for stop bits, the greater the lenience in synchronizing the different clocks, but the slower the data transmission rate.
4. **Parity** is a simple form of error checking that is used in serial communication. There are four types of parity: even, odd, marked, and spaced. The option of using no parity is also available. For even and odd parity, the serial port will set the parity bit (the last bit after the data bits) to a value to ensure that the transmission has an even or odd number of logic high bits. For example, if the data was 011, then for even parity, the parity bit would be 0 to keep the number of logic high bits even. If the parity was odd, then the parity bit would be 1, resulting in 3 logic high bits. Marked and spaced parity does not actually check the data bits, but simply sets the parity bit high for marked parity or low for spaced parity. This allows the receiving device to know the state of a bit so as to enable the device to determine if noise is corrupting the data or if the transmitting and receiving devices' clocks are out of sync.

### What is RS-232?

RS-232 (ANSI/EIA-232 Standard) is the serial connection found on IBM-compatible PCs. It is used for many purposes, such as connecting a mouse, printer, or modem, as well as industrial instrumentation. Because of improvements in line drivers and cables, applications often increase the performance of RS-232 beyond the distance and speed listed in the standard. RS-232 is limited to point-to-point connections between PC serial ports and devices. RS-232 hardware can be used for serial communication up to distances of 50 feet.

#### DB-9 Male



#### Pin Functions for RS-232

Data	
TXD (pin 3)	Serial Data Output
RXD (pin 2)	Serial Data Input
Handshake	
RTS (pin 7)	Request to Send
CTS (pin 8)	Clear to Send
DSR (pin 6)	Data Set Ready
DCD (pin 1)	Data Carrier Detect
DTR (pin 4)	Data Terminal Ready
Ground	
GND (pin 5)	Ground
Other	

RI (pin 9)	Ring Indicator
------------	----------------

## What is RS-422?

RS-422 (EIA RS-422-A Standard) is the serial connection used on Apple Macintosh computers. RS-422 uses a differential electrical signal, as opposed to unbalanced signals referenced to ground with the RS-232. Differential transmission, which uses two lines each for transmit and receive signals, results in greater noise immunity and longer distances as compared to the RS-232. These advantages make RS-422 a better fit for industrial applications.

## What Is RS-485?

RS-485 (EIA-485 Standard) is an improvement over RS-422, because it increases the number of devices from 10 to 32 and defines the electrical characteristics necessary to ensure adequate signal voltages under maximum load. With this enhanced multi-drop capability, you can create networks of devices connected to a single RS-485 serial port. The noise immunity and multi-drop capability make RS-485 the serial connection of choice in industrial applications requiring many distributed devices networked to a PC or other controller for data collection, HMI, or other operations. RS-485 is a superset of RS-422; thus, all RS-422 devices may be controlled by RS-485. RS-485 hardware may be used for serial communication with up to 4000 feet of cable.

DB-9 Male



## Pin Functions for RS-485 and RS-422

Data	
TXD+ (pin 8)	Serial Data Output (differential)
TXD- (pin 9)	Serial Data Output(differential)
RXD+ (pin 4)	Serial Data Input(differential)
RXD- (pin 5)	Serial Data Input(differential)
Handshake	
RTS+ (pin 3)	Request to Send (differential)
RTS- (pin 7)	Request to Send (differential)
CTS+ (pin 2)	Clear to Send (differential)
CTS- (pin 6)	Clear to Send (differential)
DSR (pin 6)	Data Set Ready
Ground	
GND (pin 1)	Ground

## What is Handshaking?

The method used by RS-232 for communication allows for a simple connection of three lines: TX, RX, and ground.

For the data to be transmitted, both sides have to be clocking the data at the same baud rate. Although, this method is sufficient for most applications, it is limited in being able to respond to problems such as the receiver getting overloaded. This is where serial handshaking can help. In this section we will discuss three of the most popular forms of handshaking with RS-232: Software Handshaking, Hardware Handshaking, and Xmodem.

1. **Software Handshaking:** The first form of handshaking we will discuss is software handshaking. This style uses actual data bytes as control characters, similar to the way GPIB uses command strings. The lines necessary are still the simple three line set of TX, RX, and round since the control characters are sent over the transmission line like regular data. The function SetXMode allows the user to enable or disable the use of two control characters, XON and XOFF. These characters are sent by the receiver of the data to pause the transmitter during communication.

As an example, assume that the transmitter begins to transmit data at a high baud rate. During the transmission, the receiver finds that the input buffer is becoming full due to the CPU being busy with other duties. To temporarily pause the transmission, the receiver sends XOFF, typically decimal 19 or hex 13, until the input buffer has been emptied. Once the receiver is ready for more data it sends XON, typically decimal 17 or hex 11, to resume communication. LabWindows will send XOFF when its input buffer becomes half full. In addition, in case the XOFF transmission was corrupted, LabWindows will also transmit XOFF when the buffer has reached 75% and 90% capacity. Obviously, the transmitter must also be following this protocol for it to succeed.

The biggest drawback to this method is also the most important fact to keep in mind: decimal 17 and 19 are now off limits for data values. In ASCII transmissions this typically does not matter since these values are non-character values; however, if the data is being transmitted via binary, it is very likely that these values could be transmitted as data and the transmission would fail.

2. **Hardware Handshaking:** The second method of handshaking is to use actual hardware lines. Like the TX and RX lines, the RTS/CTS and DTR/DSR lines work together with one being the output and the other the input. The first set of lines are RTS (Request to Send) and CTS (Clear to Send). When a receiver is ready for data, it will assert the RTS line indicating it is ready to receive data. This is then read by the sender at the CTS input, indicating it is clear to send the data. The next set of lines are DTR (Data Terminal Ready) and DSR (Data Set Ready). These lines are used mainly for modem communication. They allow the serial port and the modem to communicate their status. For example, when the modem is ready for data to be sent from the PC, it will assert the DTR line indicating that a connection has been made across the phone line. This is read in through the DSR line and the PC can begin to send data. The general rule of thumb is that the DTR/DSR lines are used to indicate that the system is ready for communication where the RTS/CTS lines are used for individual packets of data.

In LabWindows, the function SetCTSMode enables or disables the use of hardware handshaking. If the CTS mode is enabled, LabWindows employs the following rules:

When the PC sends data:

The RS-232 library must detect that its CTS line is high before sending data.

When the PC receives data:

If the port is opened and the input queue has room for data, the library raises RTS and DTR.

If the port's input queue is 90% full, the library lowers RTS and leaves DTR high.

If the port's input queue is nearly empty, the library raises RTS and leaves DTR high.

If the port is closed, the library lowers RTS and DTR.

3. **XModem Handshaking:** The last mode discussed here is the XModem file transfer protocol. This protocol is very common in modem communication. Although it is often used for modem

communication, the XModem protocol can be used directly between other devices if they both follow the protocol. In LabWindows, the actual implementation of XModem is hidden from the user. As long as the PC is connected to another device using XModem protocol, the LabWindows' XModem functions can be used to transfer files from one site to another. The functions are XModemConfig, XModemSend, and XModemReceive.

XModem uses a protocol based on the following parameters: start\_of\_data, end\_of\_trans, neg\_ack, ack, wait\_delay, start\_delay, max\_tries, packet\_size. These parameters need to be agreed upon by both sides, and standard XModem has a standard definition of these; however, they can be modified through the XModemConfig function in LabWindows to meet any requirement. The way that these parameters are used in XModem is by having the neg\_ack character sent by the receiver. This tells the sender that it is ready to receive data. It will try again with start\_delay time in-between each try until either it reaches max\_tries or receives start\_of\_data from the sender. If it reaches max\_tries it will inform the user that it was unable to communicate with the sender. If it does receive start\_of\_data from the sender, it will read the packet of information that follows. This packet contains the packet number, the complement of the packet number as an error check, the actual data packet of packet\_size bytes, and a checksum on the data for more error checking. After reading the data, the receiver will call wait\_delay and then send ack back to the sender. If the sender does not receive ack, it will re-send the data packet max\_tries or until it receives ack. If it never receives the ack, it informs the user that it has failed to transfer the file.

Since the data must be sent in packets of packet\_size bytes, when the last packet is sent, if there is not enough data to fill the packet, the data packet is padded with ASCII NUL (0) bytes. This can cause the received file to be larger than the original. It is also important to remember not to use XON/XOFF with the XModem protocol since the packet number from the XModem transfer is very likely to increment to the XON/OFF control character values, which would cause a breakdown in communication.

## Related Links

[KnowledgeBase 3DRAGUOY: A Quick Comparison of RS-232, RS-422, and RS-485 Serial Communication Interfaces](#)

## Legal

This tutorial (this "tutorial") was developed by National Instruments ("NI"). Although technical support of this tutorial may be made available by National Instruments, the content in this tutorial may not be completely tested and verified, and NI does not guarantee its quality in any way or that NI will continue to support this content with each new revision of related products and drivers. THIS TUTORIAL IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND AND SUBJECT TO CERTAIN RESTRICTIONS AS MORE SPECIFICALLY SET FORTH IN NI.COM'S TERMS OF USE (<http://ni.com/legal/termsofuse/unitedstates/us/>).