**Basic Serial Communications**

*The Basics*

For those new to control system programming, serial communications is often an area that is the least understood or at the very least needs some of the blanks filled in. Serial communications is currently dominated by the RS-232 interface, with RS-422 and RS-485 following close behind. There are a few basics of each serial format that should be mentioned, starting with:

**RS-232** – This is the dominant serial communication format used in the AV industry, as most manufacturers support it across their product lines. This communication format offers several advantages for the custom installer. The advantages typically include increased communication reliability, additional controls that aren’t available on the standard infrared remote and usually the ability to query the device for its current status. Some typical examples of this would be the ability to have discrete power on/off commands, as well as the ability to select discrete inputs. These functions might typically be toggled through on the handheld remote, making custom integration very difficult to be reliable.

Despite these advantages, there are some disadvantages that need to be overcome by the custom installer. RS-232 signals are based upon an unbalanced transmission, analogous to unbalanced audio, making them more susceptible to EMI and RFI. The second major disadvantage is that, by specification, RS-232 based communication is limited to a distance of 50 feet. When you add up the wiring distance used in a properly dressed rack and the distance for a cable to travel up a wall to its location and possibly back down the wall again, 50 feet of cable can be used very quickly. Fortunately, Crestron provides several methods to circumvent this type of problem. Typical solutions include:

1. Use the serial port on your Crestron processor to output data in the RS-422 format, which can travel much further and then convert the signal back to RS-232 at the destination. This method has the additional advantage of using a balanced signal which rejects EMI and RFI noise much better than RS-232.
2. Use Cresnet. You can run a Cresnet line to the item to be controlled and then simply connect a ST-COM at the device to be controlled. Cresnet can be run a cumulative distance of 5000 feet without needing any amplification; this is sufficient for all but the rarest applications. Cresnet is balanced and will also reject noise much better than RS-232.
3. Use Ethernet. With the prevalence of Ethernet in the corporate and custom home environments, you can simply add a CEN-COM to the existing LAN to provide a serial connection at any location that the LAN is available.

**RS-422** – This communication format came about as result of some of the limitations of RS-232. The biggest advantages of this format are that the distance limitation increases to 2000 feet and the transmission format changes from unbalanced to balanced. Unfortunately, most products in the AV industry do not have an RS-422 port and an outboard RS-422 to RS-232 converter must be used when talking to an RS-232 device.

**RS-485** – This communication format is another successor to RS-232 and improves upon RS-422. The distance limitation increases to 5000 feet and is also a balanced transmission. Unlike RS-232 and RS-422, this is a multi-point protocol, which means that you can have multiple devices on one RS-485 communication bus. Up to 32 devices are allowed on an RS-485 bus. When multiple devices are on a bus, they must have a unique address and the device furthest from the processor must have a terminating resistor. As with RS-422, there aren’t many devices that have built in support for this protocol and an RS-485 to RS-232 converter must be used with most AV products.

**Flow Control** – Flow control, also called handshaking, is simply a method used by two communicating devices to ensure that no data is lost due to data overruns. Any time devices communicate, there exists the possibility for one of them to send data faster than the other can process it. Flow control provides a method for the receiving device to halt the transmission of data, while data that has already been received is processed. Crestron supports both hardware and software based flow control, referred to as RTS/CTS and XON/XOFF respectively.
**Rules to keep it simple.**

Successful serial communications can be broken down into three simple categories. These are:

1. **Cabling** – Though this may seem obvious, it is one of the most overlooked items. Fortunately Crestron has provided a great tool called the Cable Database. This can be used to look up cable drawings based upon the make and model of the device or the pin out provided by the manufacturer. The Cable Database can be found by going to [www.crestron.com](http://www.crestron.com) and clicking on downloads and then going to the Cables library. If the cable you need hasn’t been placed into the Crestron Cable Database yet, there are some simple items to remember to get your cabling correct. When connecting a RS-232 device to Crestron, connect the Crestron serial ground to the devices serial ground, connect the Crestron transmit to the devices receive, and vice versa. If the device you are communicating with is using RTS/CTS handshaking, simply connect the Crestron RTS to the devices CTS and vice versa. *Note: If your device expects the DTR/DSR pins on a com port, simply short them together on the device side of the cable.*

2. **Configuration** – This category covers two areas. The first item is to ensure that serial communications are enabled. This may mean navigating through menus or setting DIP switches; very often this communication is enable by default and requires no intervention on your part. The second item is to verify that the Crestron software is programmed for the same baud rate, parity, stop and data bits as the device that you are trying to communicate with. This information can usually be found in the manual of the device that you are communicating with. If one of these items do not match you will have no communications.

3. **Protocol** – This is often the most difficult item. The protocol is simply the language that the device expects to be sent. The biggest disadvantage that the custom installer faces is that there are no standards regarding the data that will be sent to the device, other than the ASCII standard used by computers in general. This simply means that the data sent to one manufacturers device will not work with another manufacturer and often one manufacturers code will not work with another one of their own products. Lets take a look at some examples to clarify what may be expected.

**Protocols**

Most protocols in the AV industry tend to be simple, using an ASCII based protocol that can be typed in a terminal program to issue commands to the device. We’ll use an Extron switcher as an example: to switch to input one, we would simply send: 1! This information was obtained by referring to the technical manual that accompanied the switcher and reading the section on RS-232 based control. To make an Extron matrix switcher connect input 1 to output 3 we would send: 1*3! This is an example of an extremely easy to use and integrator friendly protocol.

Once the format of the serial data is known, it is simply a matter of entering that data into the com port definition in SIMPL Windows and triggering the data at the proper time.
In the above example, two data bytes are being transmitted to switch the device to input 1. The bytes transmitted are ASCII characters 1 and !.

**ASCII vs. Hexadecimal values**

All serial data comes out of the com port as a series of binary values and every ASCII character has an equivalent binary or hexadecimal value. The numeric 1 on your keyboard doesn’t actually have a value of 1; it has a value of 31 hexadecimal. It is important to note that any items typed into the com port definition will represent the ASCII values being transmitted, unless special steps are taken. When a serial protocol specifies a numeric value to be transmitted, it must be determined if the value is the ASCII character or if the value is a decimal or hexadecimal. If it is an ASCII value, we simply type it into the com port as shown in the above example. If the numeric value represents something other than ASCII, we will convert the value to hexadecimal and transmit it.

In the above example we are sending a combination of hex values and ASCII characters. This example starts by sending a hex 02 followed by the ASCII characters PON followed by hex 03. After reading the serial protocol, it was determined that the 2 and 3 specified are hexadecimal values. To send any hexadecimal value we simply use the format of: \xYY, where YY represents the hex value to be transmitted. Using this information, we can send out any hexadecimal value that is needed. In a similar fashion, we can transmit any ASCII character as its hexadecimal equivalent. In the first example we sent 1! to switch to input 1. The following hexadecimal values would do the same job: \x31\x21 .

*NOTE: You may see hexadecimal values represented in several different fashions. Some examples of this are: \x01, <01>, 01x01, 0x01 or 01h. All of these simply represent one hexadecimal.*

We use the \x to tell SIMPL Windows that we wish to substitute a hex value for the ASCII characters typed. There are two other character substitutions that are useful to know; these are: \r and \n. The \r substitution is used to tell SIMPL Windows that we wish to transmit a Carriage Return, while the \n is used to send a Carriage Return followed by a Line Feed. These are typically represented as \<CR> or \<CR><LF> in a manufacturers documentation.

**Troubleshooting the serial connection**

You’ve wired the serial connection, configured the baud rates, and have programmed the data into the control system and it doesn’t work! What to do? The first step in troubleshooting serial communications is to review everything and verify your cabling, device configuration, and look for typos in your serial data. If everything still looks good, but it isn’t working, we move to the next level and start to substitute known good elements. Let’s take this in several different steps to isolate our problem areas.
1. Does the device communicate outside of the control system? If the manufacturer supplies any type of communication/configuration software and cabling, hook it up and verify that your device operates properly. If the device doesn’t work with the manufacturer’s software and cable it won’t work with the control system.

2. Assuming that the device worked in step 1, substitute the known good and working cable with the one installed in your system. If the manufacturers software stops communicating with the device, then you have a problem with the cabling; correct it and retest. If it works properly then you know your installed cable is good.

3. With the cabling resolved or determined to be working, look at the manufacturer’s software to see what the working baud settings are. Verify that your control program is using these same settings. If the software doesn’t display these settings, call the manufacturer and ask or check the manual to see if it is specified.

4. With the cabling and baud settings resolved, we can now try and see if the control system is controlling the device. If the control system doesn’t control the device then we need to look at the serial data being sent out of the control system. The best way to resolve this is with a working, known good data string. Ask the manufacturer to provide you a known good data example for a function that is easy to detect, such as power on or power off. Compare this serial data to what you entered and correct anything that is wrong. Usually a good example will hi-light the error that has been made and allow you to correct the serial data. Take special care to find out where the manufacturer expects ASCII numeric characters vs. actual hexadecimal values.