

Using Revolution for Process Control

Courtesy of Sarah Reichelt

Communication Basics

Revolution cannot talk directly to USB ports, but if you connect a USB-serial adapter to a USB port and install the drivers, Revolution sees that device as a virtual serial port. Then you can communicate through that USB port using Revolution's serial commands.



I use two types of adapters: Keyspan and FTDI. The Keyspan High Speed serial adapter <http://www.keyspan.com/products/usa19hs/> has a standard D9 serial connection, while the Keyspan Twin Serial adapter <http://www.keyspan.com/products/usa28xg/> handles the old Mac-style MiniDin connections. For FTDI, I use one of the Easy-Sync adapters <http://www.easysync-ltd.com/index.html?lang=en-uk&target=d12.html> which also provide a D9 connection.

For testing, I have a serial test stack <http://www.troz.net/Rev/tutorials/SerialTest.rev.gz> which allows basic text communications. Even if no hardware is connected to your adapter, you can still use this to confirm that the port is recognised. If the adapter has indicator lights, you can also check that sending works OK.

With Mac OS X, when you install the drivers and connect one of these adapters, it can be detected using "the driverNames". This returns a list of detected ports, one per line. The third item of each line is the name that you use to refer to the serial port in all other commands.

For Windows, Revolution allows you to specify "COM1:", "COM2:" etc., up to "COM9:". After connecting the serial device, you can use the Control Panel to find which COM port it has been allocated to.

To use the port, you first open it. The serialControlString specifies the settings required for your hardware. Supposedly, you can use either "open file" or "open driver", but I have got the best results using "open file" on Windows systems and "open driver" on Mac OS X. However it has been some time since I last tested this.

When opening the port, you have to say what sort of data you will be sending and receiving (text or binary) and whether you are going to read, write, append or update. I find the most flexible system is to use binary update as this seems to work for anything.

Once the port is open, you can use the "read" and "write" commands to receive and send data. Some devices require specific line endings to detect when a command has been received and to indicate the end of a message, so this may require some experimentation. If the device you are talking to accepts text, then it is easy to send strings. If the device requires binary data, you will have to convert each character to its numeric equivalent using `charToNum()` before sending.

Finally and extremely importantly, you must close the port when finished. If you have a standalone and you quit without closing the serial port, not only will the process not quit completely, but it is likely that no other application will be able to access that port.

Using Revolution for Process Control

Courtesy of Sarah Reichelt

Those are the basics of serial communications. There are a lot more details, mostly to do with handling when things go wrong: being unable to open the port, not getting any response, getting the wrong response etc., but since these will vary with your application, that's something for everyone to work out for themselves.

Real-World Applications: Process Control

I use these techniques for process control. We run bacterial fermenters and other processes, required controlled temperature, pH, oxygen etc. Using two main types of hardware, we can monitor and control all these from Revolution. This requires hardware to translate the reading from various sensors into the correct electrical signals and then hardware to allow this data to be available to the computer.

For temperature data, we use Dallas temperature sensors <http://www.maxim-ic.com/quick_view2.cfm/qv_pk/2812> which have the advantage that they require no calibration and return an actual temperature reading. Our electronics guru built a custom box for connecting up to 255 of these sensors in a network, and reading them using text commands. Here is a screen shot of the Revolution program that reads these sensors and monitors them for any alarm conditions.

For all other sensors and for controllers (valves, pumps, heaters, chillers), we use a selection of Weeder modules <<http://www.weedtech.com/>>. Again, these communicate using text over the serial port. They can be connected in a chain, so multiple modules can all be controlled by the same application. The input sensors need to be providing the correct level of input

ID	Name	Value	Low alarm	High alarm
35	BF freezer	-19.4		0.0
34	BF fridge	2.2		12.0
37	BTK Culture Tank 1 T	21.2		
38	BTK Culture Tank 2 T	21.4		
36	BTK1 Air exit T	61.5		
17	BTK1 Chiller	23.1		
18	BTK1 Fermenter	25.9		
6	BTK1 Glucose T°	23.5		
41	BTK2 Fermenter T	23.6		
42	BTK2 Glucose T	23.3		
12	Cold Room Small	6.2		12.0
39	Dalek2 Air Exit	60.6		
19	Dalek2 Fermenter	5.6		
20	Dalek2 Glucose	20.9		
40	Dalek2 Harvest Drum	21.9		
30	Dalek2 Hot Jacket	20.9		
9	Drying Room Silac	25.4		
1	Incubator 30°	42.4		
8	Incubator Room	28.9	20.0	40.0
21	Pressure cooker	22.2		
23	Silac Chemical Store	18.6		25.0
14	Silac Room	16.2		23.0
16	Silac Room - FD Blue	17.7		
15	Silac Room - FD White	37.4	25.0	45.0
25	Tank T1 Mixer T°	22.3		

(normally 0 - 5V) so depending on the sensors being used, this may require an extra piece of hardware to convert the signal. Here (to the left) is a screen shot of the application communicating with this. It has different groups, so to add a new module, you just duplicate the appropriate group, change it's label and away it goes.

The screenshot shows the 'Weeder Communications' interface. It features a grid of modules organized into five input modules (A-E) and two analog modules (M-N). Each module has a 'Module active' checkbox and a 'Read now' button. The modules are configured with various sensors and controllers, such as 'Dalek2 Oxygen solenoid', 'BTK2 Chiller solenoid', and 'Formulation Alkali solenoid'. The interface also includes a 'Comms log' on the right side, showing a list of communication messages between the software and the hardware modules.

The top row of groups shows 5 input modules i.e. modules that read data from various sensors. This is returned

Using Revolution for Process Control

Courtesy of Sarah Reichelt

as a number which needs to be converted to get a true reading. The bottom row has three digital output modules and two analog output modules. The difference is that digital outputs are either on or off. Analog outputs can be partly on - this works best for proportional valves that may only need to be opened a small amount.

These two applications do all the communications with the hardware, although there are multiple processes running, controlled by multiple computers. The other computers all communicate with these two applications using shared text files and either get data or send control messages.