Serial Console

Serial Console

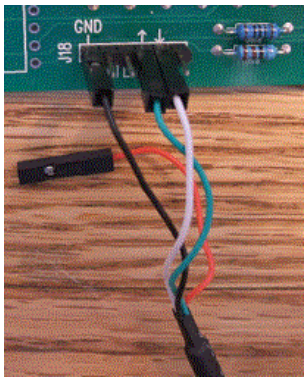
The Relay Trainer has a serial console port. A USB to TTL serial cable using either the Prolific or FTDI conversion chips may be used.

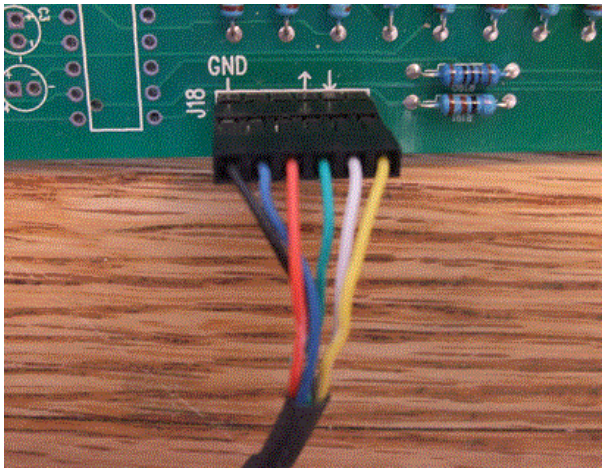
The Windows driver for Prolific PL2303-based cables can be found here.

The Windows driver for FTDI-based cables can be found here.

I've found that the combination of these drivers and Windows (particularly 64-bit Windows-7) is somewhat buggy. If the you find that your system hangs or that you get BSOD, try plugging the cable into a different USB slot, or through a USB hub. I've generally not had problems with the open-source drivers provided with Linux.

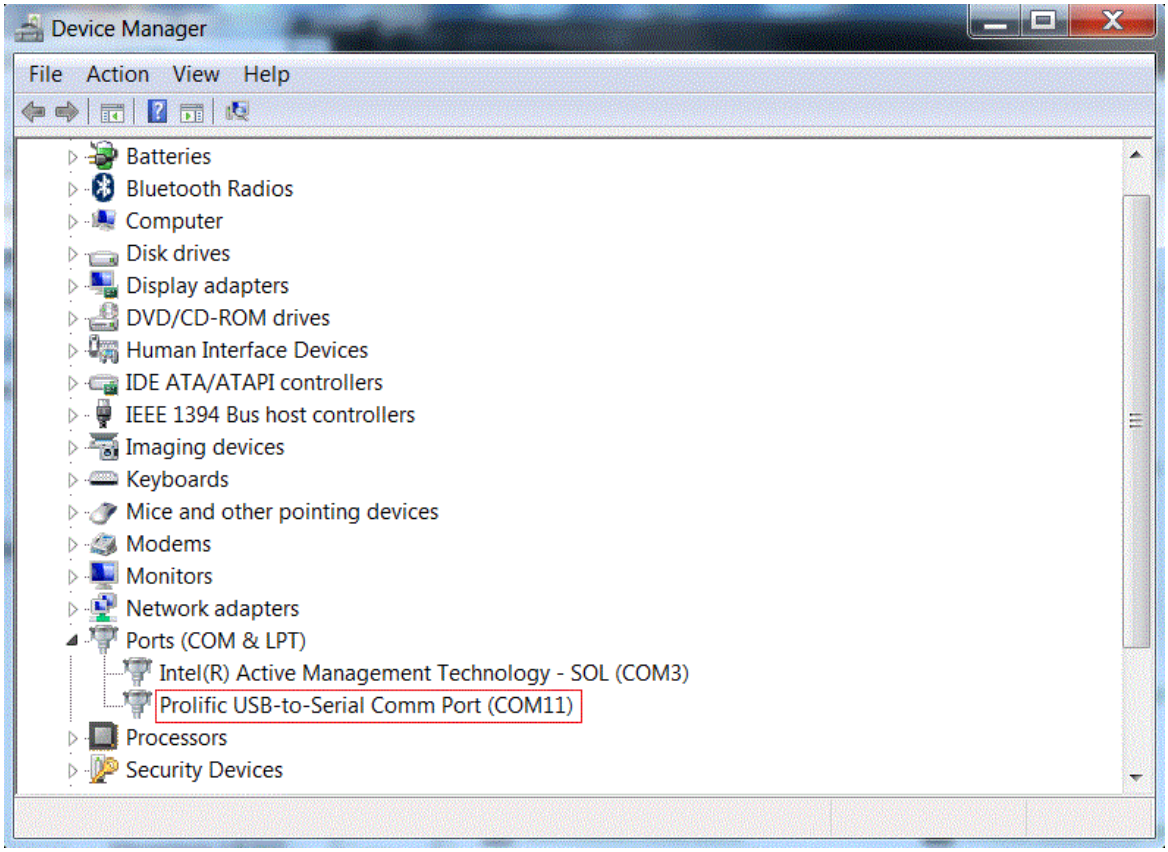
There are several styles of connectors available on these cables. This is how to connect them to the relay computer:



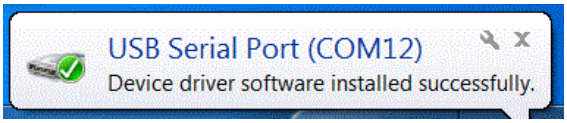


I recommend that you use PuTTY as the terminal emulator for Windows, or Minicom for Linux.

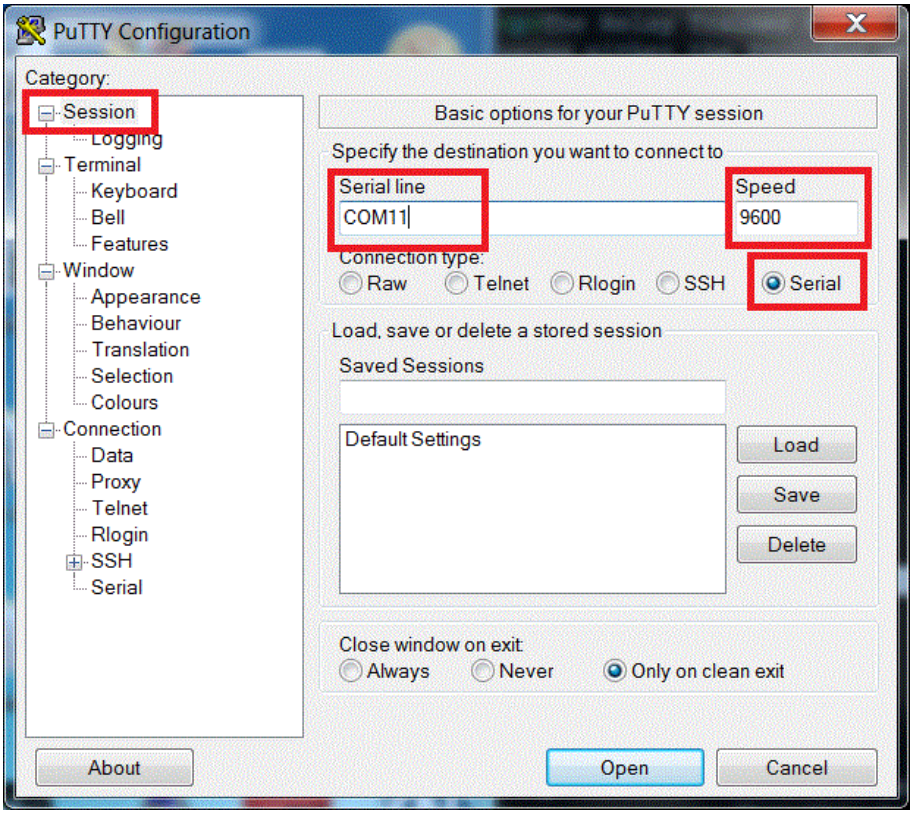
Unfortunately, PuTTY does not provide a drop-down list of available COM ports, so you have to find the COM port for the USB to TTL serial conversion cable by looking at Control Panel => Device Manager



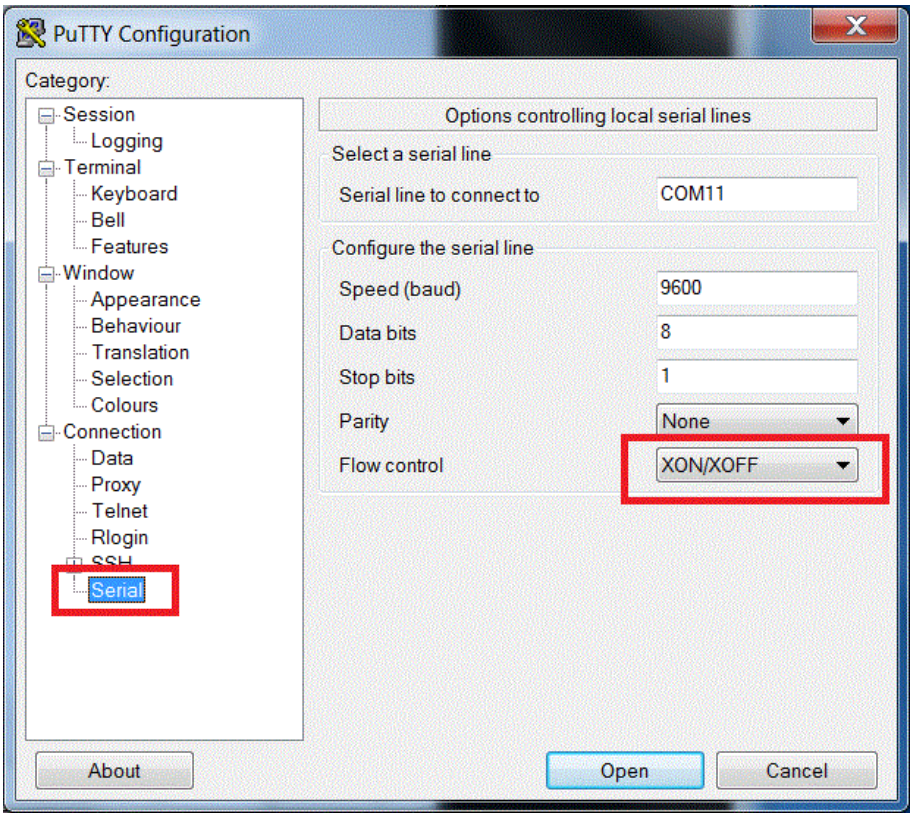
Although the very first time you plug the USB to serial cable into the machine you'll see a balloon tip with the COM port:



When you start PuTTY, its configuration screen is shown. Make sure "Session" is selected on the left side, if you don't see this screen. It should be configured as follows. Select "Serial" for connection type, then type in the COM port of the USB to TTL serial cable and leave the BAUD rate set at 9600.



Under serial settings (select on the left), select XON/XOFF flow control. Finally click Open to start the terminal emulator.



Once the cable is connected and the Relay Trainer is powered on, you should see the following on the terminal emulator. Type 'h' to get a list of commands:

```
Relay Computer READY

>

>h
help            Show this help text
```

```
h               Show this help text
s               Step one instruction
g [hh]          Go (at current or address hh)
j hh            Jump to address hh
d               Hex dump
r               Show registers
speed h         Set speed 0 - F
clear           Clear memory
save            Save memory to EEPROM
load            Load memory from EEPROM
t [on|off]      Turn tracing on / off
b [hhhh]        Set/Clear breakpoint
a hhhh          Assemble
clr             Clear symbol table
sy              Show symbol table
u hhhh          Unassemble
v expr          Evaluate expression
aa: hh hh ...   Load memory with hex data

>
```

Here are some of the things which can be done through the serial port:

Show hexadecimal dump of the memory:

```
>d
00: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
08: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
10: 48009000 4800e901 0062011a 08000002 08e00102 00660218 08e00100 4018ff12
18: 08e00001 4018ff12 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
20: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
28: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
30: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
38: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
40: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
48: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
50: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
58: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
60: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
68: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
70: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
78: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
80: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
88: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
90: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
98: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
a0: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
a8: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
b0: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
b8: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
c0: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
c8: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
d0: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
d8: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
e0: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
e8: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
f0: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00
f8: c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00 c810ff00

>
```

Show the registers:

```
>r
From relays:
  PC = 00
  Write data = ff
To relays:
  Current insn = c810ff00
  A data = ff
  B data = 00
Current speed = 5

>
```

Use the built-in assembler to enter an a program from assembly language source code:

```
>a 40
40:  st #-8, 0
41: loop incjne 0, loop

Adding fixup
```

```
Undefined symbol
Fixup at 41
42:  halt
43:
```

Note: hit Ctrl-C to exit the assembler. You should type **sy** to verify that there are no remaining undefined symbols after assembly is complete. Use the **clr** command to clear the symbol table in preparation for assembling a new program.

Use the built-in dis-assembler to check the program just entered:

```
>u 40
40  4800_f800         st      #0xf8, 0x00
41  802a_0041         incjne  0x00, 0x41
42  c810_ff00         halt
43  c810_ff00         halt
44  c810_ff00         halt
45  c810_ff00         halt
46  c810_ff00         halt
47  c810_ff00         halt
48  c810_ff00         halt
49  c810_ff00         halt
4a  c810_ff00         halt
4b  c810_ff00         halt
4c  c810_ff00         halt
4d  c810_ff00         halt
4e  c810_ff00         halt
4f  c810_ff00         halt
50  c810_ff00         halt
51  c810_ff00         halt
52  c810_ff00         halt
53  c810_ff00         halt
54  c810_ff00         halt
55  c810_ff00         halt

>
```

The **sy** command can be used to show the symbol table used by the assembler:

```
>sy
loop = 41
Symbol table size b
```

There are only 256 bytes available for the symbol table. The built-in assembler is a one-pass assembler, which means forward references also take up space in the symbol table.

Instruction tracing can be enabled, so that you can watch what the relay computer is doing while a program is running:

```
>t on

>g 40
40  4800_f800         st      #0xf8, 0x00  B[00] <- f8
41  802a_0041         incjne  0x00, 0x41   A[00] <- f9  PC <- 41
41  802a_0041         incjne  0x00, 0x41   A[00] <- fa  PC <- 41
41  802a_0041         incjne  0x00, 0x41   A[00] <- fb  PC <- 41
41  802a_0041         incjne  0x00, 0x41   A[00] <- fc  PC <- 41
41  802a_0041         incjne  0x00, 0x41   A[00] <- fd  PC <- 41
41  802a_0041         incjne  0x00, 0x41   A[00] <- fe  PC <- 41
41  802a_0041         incjne  0x00, 0x41   A[00] <- ff  PC <- 41
41  802a_0041         incjne  0x00, 0x41   A[00] <- 00

>
```

You can select a hex-dump from a program listing and paste it into the terminal emulator to load an assembled program into the memory- in this case the LFSR example program:

```
>00: 00000001

>02: 00000009

>03: 00000080

>05: 84081d10

>06: 802a0305

>07: c810ff00

>10: 08000001

>11: 08900101
```

```
>12: 08900101

>13: 08900101

>14: 49800301

>15: 0062011a

>16: 48e00101

>17: 0062011c

>18: 48e00101

>19: 0062011c

>1a: 08a00000

>1b: 4018ff1d

>1c: 08800000

>1d: 4018ff00

>
```

Then execute it:

```
>g 5
05  8408_1d10        jsr     0x1d, 0x10   A[1d] <- 06  PC <- 10
10  0800_0001        st      0x00, 0x01   B[01] <- 01
11  0890_0101        rol     0x01         B[01] <- 02
12  0890_0101        rol     0x01         B[01] <- 04
13  0890_0101        rol     0x01         B[01] <- 08
14  4980_0301        andto   #0x03, 0x01  B[01] <- 00
15  0062_011a        jeq     0x01, 0x1a                PC <- 1a
1a  08a0_0000        asl     0x00         B[00] <- 03
1b  4018_ff1d        jmp     0x1d                      PC <- 1d
1d  4018_ff06        jmp     0x06                      PC <- 06
06  802a_0305        incjne  0x03, 0x05   A[03] <- 81  PC <- 05
05  8408_1d10        jsr     0x1d, 0x10   A[1d] <- 06  PC <- 10
10  0800_0001        st      0x00, 0x01   B[01] <- 03
11  0890_0101        rol     0x01         B[01] <- 06
12  0890_0101        rol     0x01         B[01] <- 0c
13  0890_0101        rol     0x01         B[01] <- 18
14  4980_0301        andto   #0x03, 0x01  B[01] <- 00
15  0062_011a        jeq     0x01, 0x1a                PC <- 1a
1a  08a0_0000        asl     0x00         B[00] <- 07
1b  4018_ff1d        jmp     0x1d                      PC <- 1d
1d  4018_ff06        jmp     0x06                      PC <- 06
06  802a_0305        incjne  0x03, 0x05   A[03] <- 82  PC <- 05
05  8408_1d10        jsr     0x1d, 0x10   A[1d] <- 06  PC <- 10
10  0800_0001        st      0x00, 0x01   B[01] <- 07
11  0890_0101        rol     0x01         B[01] <- 0e
12  0890_0101        rol     0x01         B[01] <- 1c
13  0890_0101        rol     0x01         B[01] <- 38
14  4980_0301        andto   #0x03, 0x01  B[01] <- 00
15  0062_011a        jeq     0x01, 0x1a                PC <- 1a
1a  08a0_0000        asl     0x00         B[00] <- 0f
1b  4018_ff1d        jmp     0x1d                      PC <- 1d
1d  4018_ff06        jmp     0x06                      PC <- 06
06  802a_0305        incjne  0x03, 0x05   A[03] <- 83  PC <- 05
05  8408_1d10        jsr     0x1d, 0x10   A[1d] <- 06  PC <- 10
10  0800_0001        st      0x00, 0x01   B[01] <- 0f
11  0890_0101        rol     0x01         B[01] <- 1e
12  0890_0101        rol     0x01         B[01] <- 3c
13  0890_0101        rol     0x01         B[01] <- 78

>
```