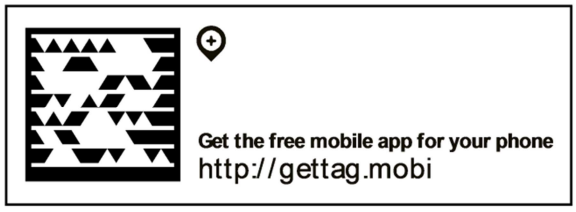
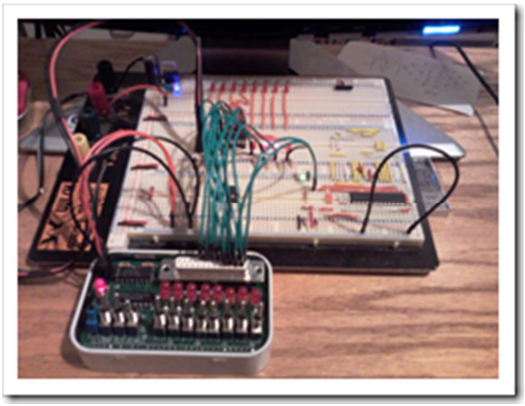
Serial Port Adapter for Membership Card

Chuck Bigham
10/17/2010

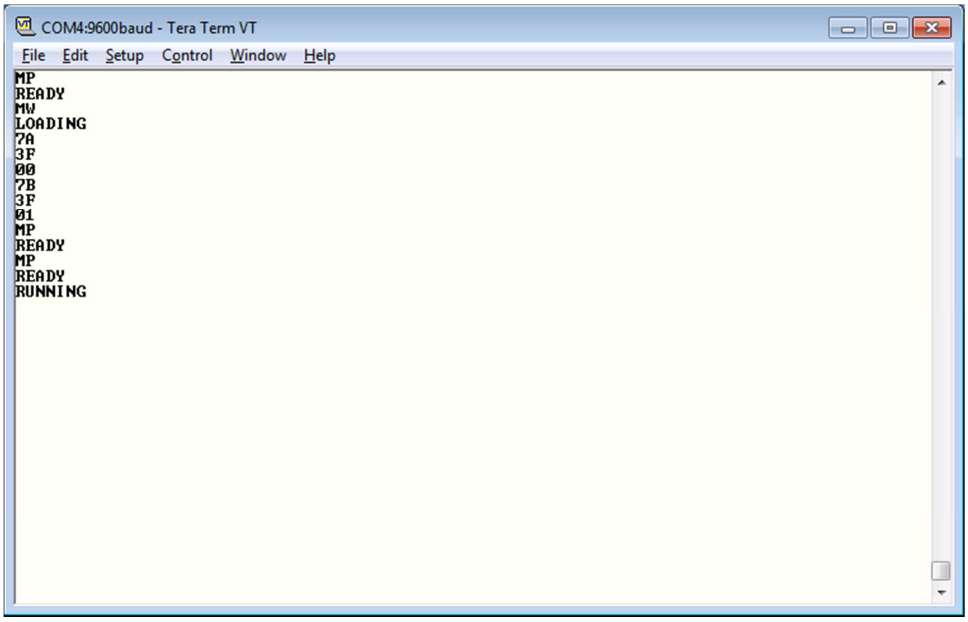


Serial Port Adapter for Membership Card



The Membership Card has a 25-pin control port that Lee intended to be used with a PC parallel port to control the Membership Card. Unfortunately, many new PCs, especially laptops, do not have a parallel port; or the PC runs Windows Vista or Windows 7, two operating systems that do not provide easy access to the parallel port for user programs. These two issues will prevent many people from using a PC control program for the Membership Card.

I decided to solve the first problem by creating a hardware RS232 to Membership Card adapter that can be used by any serial port-aware application – from a simple terminal program to a purpose-built front end. I also wrote a Windows program to use a serial port talk to the adapter, one that a person using an operating system that enables access to the parallel port could write a driver for and "plug-in" to the application, but I have neither the hardware nor the software required to create such a driver.



Serial Port Adapter

My serial port adapter is based on a Picaxe® 20X2 from Revolution Education. The Picaxe is a PIC-based microcontroller that is programmed in BASIC over a serial connection using a free IDE environment. The 20X2 is overkill for this application in some respects, I'm using only a quarter of the available program memory, but it does offer a bi-directional 8-bit port as well as sufficient other input and output lines to control the Membership Card and to communicate with the PC.

I considered using other interfaces, including other Picaxe chips, but the 20X2 gives me a flexible one-chip solution to connecting to the Membership Card. All I need is the 20X2, one resistor, a 9-pin and a

25-pin connector and a 3 to 5-volt power source and we're ready to go. I have tested the adapter with both a standard 9-pin serial port and a USB-to-serial converter with no problems.

The converter schematic is simple, connecting the input and output pins on the Picaxe to the DB9 serial connector from the PC and to the DB25 connector on the Membership Card. The schematic as drawn includes a standard Picaxe enhanced download circuit – the serial connection to the computer also functions as a download circuit for the Picaxe enabling it to be re-programmed in-circuit. It also includes four LEDs to indicate the state of the Membership Card, these LEDs and their associated resistors can be left off the converter if you desire. The schematic and parts list are in Appendix A.

The software for the 20X2 is a simple state machine that listens for commands and data from the PC on the serial port and then sends those commands and data to the membership card via the 25-pin interface. Due to the way the Picaxe software works, each command is a two-character command. After the 20X2 has processed the command, it sends a response back to the PC over the serial connection. Each response is followed by a CRLF pair so that the PC can read each response as a line from the serial port. The commands that the 20X2 listens for are listed in the following table. All commands and data sent to the adapter must be in upper-case only and sent without a following CRLF pair, and any other command sent to the 20X2 is ignored.

Command	Response	Meaning
RE	RESET	Put the Membership Card in RESET mode.
LD	LOADING	Put the Membership Card in LOAD mode.
GO	RUNNING	Put the Membership Card in RUN mode.
MP	MP	Set Memory Protect to read-only.
MW	MW	Set Memory Protect to read-write.
ID	ID	Input down.
IU	IU	Input up

All string sent to the 20X2 must be two bytes long – when a Picaxe chip executes a receive serial data operation it will block on the operation until it receives the exact number of bytes specified in the operation. To ensure that the 20X2 never blocks waiting for serial data, and that it does not get out of sync with the controlling program, all commands and data sent to the adapter are two-byte strings. The Picaxe is not a particularly fast processor; I've found that I need to insert a 125ms delay after each character to enable the 20X2 to keep up with a steady flow of data from the PC.

When the Membership Card is in LOAD or RUN mode the adapter will accept and send bytes formatted as hex strings to the Membership Card. The adapter converts the hex string to a byte value and then sets the data lines to the specified byte. Giving the responsibility to the adapter to convert strings to byte values enables text-mode serial terminals to control the Membership Card. For example, the following string could be sent to the Membership Card from a terminal program to load a simple program into the Membership Card and then run it.

```
RELDMW7A3F007B3F01REGO
```

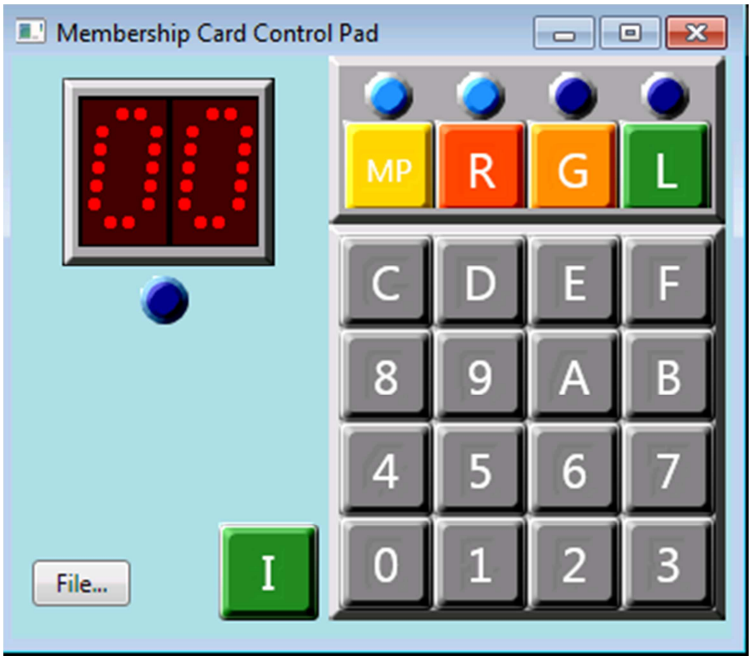
Note that the program data (`7A3F007B3F01`) is sent as hex strings without separating them with a press and release of the input button. In LOAD mode the adapter will strobe the INP line each time a byte is sent to the adapter, making it easier to send data to the Membership Card from terminal programs. Once the application is running, the Q LED should turn on every time you send the string "ID" to the adapter, and should turn off when you send "IU".

After each byte is sent to the Membership Card the data bus is read and the byte is converted into a hex string and sent to the PC followed by a CRLF pair.

The BASIC source code for the 20X2 is included in Appendix B. A version documented with extensive source code comments is included in the distribution zip archive.

Membership Card Control Panel

My Windows application for controlling the Membership Card is called, with very little imagination, the Membership Card Control Panel. The application provides a simple interface to the Membership Card that will be familiar with any person who has used an ELF-class computer or emulator. It has a hex keypad, a two-digit hex display, four command keys for setting the Membership Card operation mode, and an input button.



The control panel also includes an indicator for the mode that the Membership Card is in and a button to open a file in Intel HEX format and load that file on the Membership Card.

The control panel uses “COM4” by default. If you need to use a different port, open the installation directory (ProgramFiles\BramblyHill\MCControlPad is the default) and edit the text file called “SerialPort.txt” to change the port name in the file to the one that you want to use.

Running the Membership Card from the control panel is straightforward. Connect the Membership Card to the serial port adapter, connect the adapter to the PC, and start the Membership Card Control Panel. The gestures needed to load and run a simple program are:

- Press R to reset the Membership Card.
- Press L to put the Membership Card in LOAD mode.
- Press MP to set the memory to read-write.
- Press "7A" to send the first byte.
- Press "3F" to send the second byte.
- Press "00" to send the third byte.
- Press "7B3001" to send the rest of the bytes.
- Press R to reset the Membership Card.
- Press G to run the program.
- Press I to turn on the Q LED, release I to turn the Q LED off.

As with direct control of the Membership Card with a terminal, it is not necessary to press INP between each byte that you are sending to the Membership Card. After each byte is loaded, the hex display is updated with the value on the Membership Card data bus.

Once the program is running, the Q LED on the Membership Card should turn on when you press the input button on the Membership Card Control Panel, and turn off when you release the input button. You can also press and hold the input button on the Membership card to illuminate the Q LED.

The mode indicator will show one of the following.

Display	Membership Card state
RESET	RESET mode.
READING	LOAD mode with memory set read-only.
LOADING	LOAD mode with memory set read-write.
RUNNING	RUN mode.

The **File…** button opens a standard Open File dialog box that you can use to open an Intel HEX format file. After pressing OK, the control panel puts the Membership Card in LOAD mode with memory set to read-write, and then sends each data byte from the file to the membership card. At the end of the data the Membership Card is reset and memory set to read-only.

The control panel only does rudimentary validation that the file is in Intel HEX format. It only sends data to the Membership Card if the line starts with a color (":") and has a zero ("0") as the eighth character in the line. The control panel then sends all data to the Membership Card, starting with the ninth character and ending two characters before the end of the line, ignoring the CRC value at the end of the line.

I've tested file sending with files from the A18 assembler and the example Intel HEX file from Wikipedia.

Getting the software

The compiled software is available from the retrocomputing Web site as a Windows installer. The Membership Card Control Panel requires the Microsoft .NET Framework 2.0 or later, if you're running Windows NT or later you probably already have it installed, 3.0 is standard with Windows Vista, and 3.5 is standard with Windows 7. If you need to, you can download the .NET Framework from Microsoft at http://www.microsoft.com/net/download.aspx. The current version on the Microsoft Web site is version 4.0.

You can download a Microsoft Visual Studio 2008 project from retrocomputing and modify the code under the Creative Commons 3.0 Non-Commercial Share and Share Alike license. If you don't have Visual Studio, you can download the free Visual Studio C# express from Microsoft at http://www.microsoft.com/express/Downloads/#2010-Visual-CS.

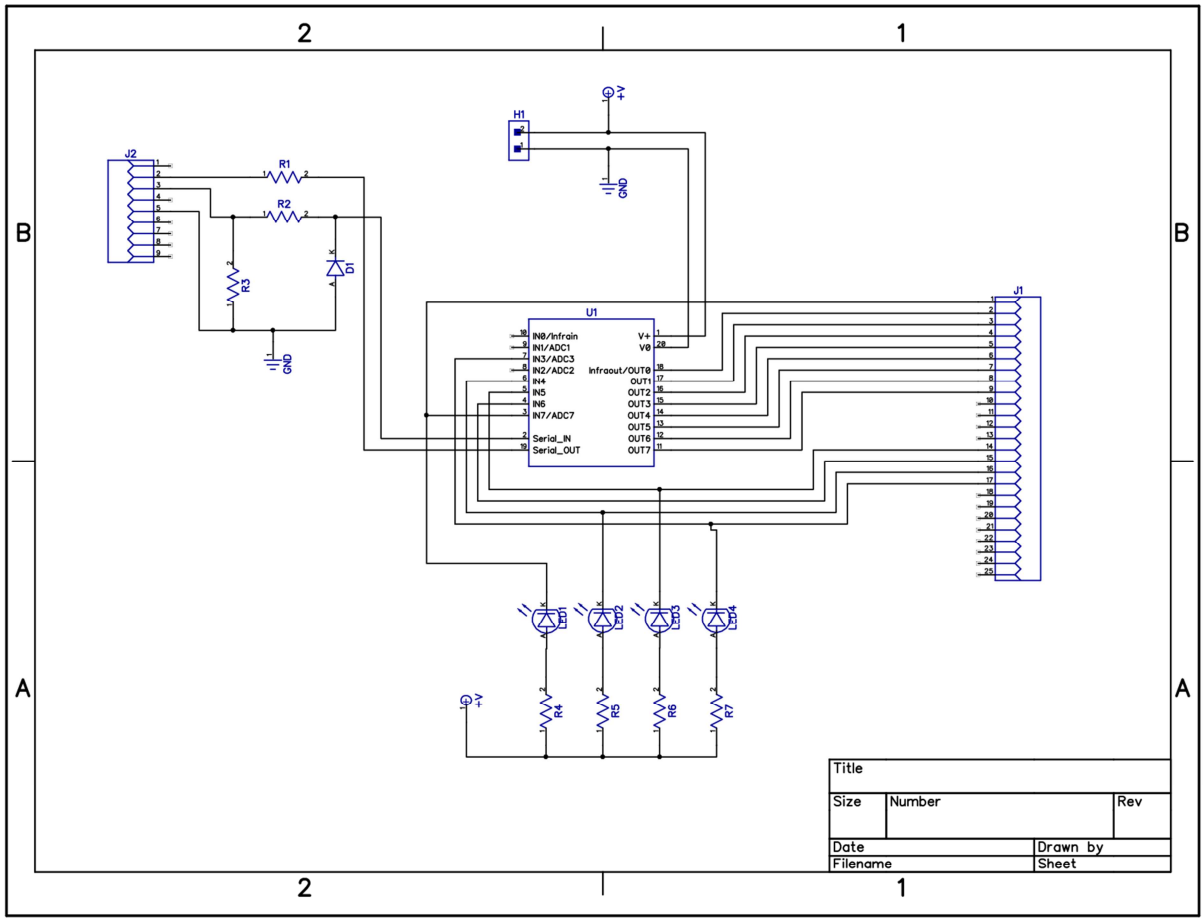
The Picaxe BASIC program for the 20X2 is included in the zip file with the installer. The Picaxe programming editor is available from Revolution Education (http://www.rev-ed.co.uk/picaxe/). You can modify the Picaxe code under the same Creative Commons 3.0 Non-Commercial Share and Share Alike license.

The hex display control is based on the **DoubleBufferedControl** in the GPS.NET 3.0 toolkit available from CodePlex. GPS.NET 3.0 is released under the LGPL – since I'm not sure how the two licenses work together, I will not be distributing the source for the **DoubleBufferedControl** along with my code. If you want to recompile the application you'll need to download GPS.NET 3.0 from CodePlex at http://gps3.codeplex.com/ and make the source code library available at build time.

If you are interested in creating a parallel port driver for the Membership Card Control Panel, or if you have improvements that you would like to see in the main distribution, please feel free to send them to me, chuck@bramblyhill.com.

Appendix A – Schematic and Parts List

A PDF version of the schematic is included in the distribution zip archive.



Parts list

U1	Picaxe 20X2
J1	DB25F
J2	DB9F or other connector for serial input. The original uses the Picaxe serial programming cable.
H1	2-pin connector or other power supply connector. The original was built on a breadboard, so I just plugged a battery into the power supply.
D1	BAT-85 small signal diode
LED1 – LED4	Light-emitting diode. The original uses red T-1 LEDs, but you can use whatever you have on hand.
R1	220-ohm ¼ watt
R2	22K-ohm ¼ watt
R3	10K-ohm ¼ watt
R4-R7	470-ohm ¼ watt (or your favorite LED dropping resistor for 5 volts.)

The 20X2 is available from several vendors. You can use your favorite search engine[1] to find a source.

[1] I suggest bing.com, but then I work for Microsoft and would.

Appendix B

Source code for the Membership Card converter.

```
' ---- [ MC Interface ]-------------------
' File..... MCInterface.bas
' Purpose... Interface between Membership Card and PC
' Author.... Chuck Bigham
' E-Mail.... chuck@bramblyhill.com
' WWW....... http://www.bramblyhill.com
' Started... 01 October 2010
' Updated...

' Target PICAXE...
#PICAXE 20X2

' ---- [ Program Description ] ------------
' Interface between a serial only PC, like
' my laptop, and Lee Hart's Membership Card
' ELF class computer.

' ---- [ Revision History ] ---------------
' 2010-10-01: First draft of application.
' 2010-10-17: Refactoring and clean up.

' ---- [ Constants ] ----------------------
SYMBOL DefaultC  = %00001010
SYMBOL ModeRESET = 0
SYMBOL ModeLOAD  = 1
SYMBOL ModeRUN   = 2

' ---- [ Variables ] ----------------------
SYMBOL outByte = B5
SYMBOL inByte  = B6

SYMBOL index      = B7
SYMBOL highNibble = B8
SYMBOL lowNibble  = B9
SYMBOL hexByte    = B10

SYMBOL elfMode    = B11
SYMBOL inpPause   = B12

SYMBOL counter = W27

SYMBOL MP    = C.0    ; SELECTIN -- active LOW
```

```
SYMBOL LOAD  = C.1    ; AUTOFEED -- active LOW
SYMBOL CLR   = C.2    ; INIT     -- active HIGH
SYMBOL INP   = C.3    ; STROBE   -- active LOW

' ---- [ EEPROM Data ] --------------------

DATA 00, (48,49,50,51,52,53,54,55,56,57,65,66,67,68,69,70)

' ---- [ Initialization ] -----------------

PowerOnReset:
     dirsB = %11111111
     dirsC = %00001111

     PinsC = DefaultC

     inpPause = 100

     WAIT 2

     GOSUB ResetMode

' ---- [ Main Code ] ----------------------

MainLoop:

     SERRXD HighNibble, LowNibble

     IF HighNibble = "R" THEN
          GOSUB ResetMode
     ELSE IF HighNibble = "L" THEN
          GOSUB LoadMode
     ELSE IF HighNibble = "G" THEN
          GOSUB RunMode
     ELSE IF HighNibble = "M" THEN
          GOSUB MemoryProtect
     ELSE IF HighNibble = "I" THEN
          GOSUB InputSwitch
     ELSE
          GOSUB HexToBinary

          outByte = HighNibble + LowNibble
          outpinsB = outByte

          TOGGLE INP
```

```
            PAUSE inpPause
            TOGGLE INP

            GOSUB WriteData
      ENDIF

      GOTO MainLoop


' ---- [ Subroutines ] --------------------

ResetMode:
      pinsC = DefaultC
      elfMode = ModeRESET
      SERTXD ("RESET",CR,LF)
      SERTXD ("MP",CR,LF)
      RETURN

LoadMode:
      IF elfMode = ModeRUN THEN
            GOSUB ResetMode
      ENDIF

      LOW LOAD
      elfMode = ModeLOAD
      SERTXD ("LOADING",CR,LF)
      RETURN

RunMode:
      IF elfMode = ModeLOAD THEN
            GOSUB ResetMode
      ENDIF

      HIGH CLR
      elfMode = ModeRUN
      SERTXD ("RUNNING",CR,LF)
      RETURN

MemoryProtect:
      IF LowNibble = "W" THEN
            HIGH MP
            SERTXD ("MW",CR,LF)
      ELSE
            LOW MP
            SERTXD ("MP",CR,LF)
```

```
     ENDIF
     RETURN

InputSwitch:
     IF LowNibble = "D" THEN
          LOW INP
          SERTXD ("ID",CR,LF)
     ELSE
          HIGH INP
          SERTXD ("IU",CR,LF)
     ENDIF
     RETURN

HEXToBinary:

     hexByte = HighNibble
     GOSUB DoHexToBinary
     HighNibble = hexByte << 4

     hexByte = LowNibble
     GOSUB DoHexToBinary
     LowNibble = hexByte

     RETURN

DoHexToBinary:
     IF hexByte >= 65 THEN
          hexByte = hexByte - 55
     ELSE
          hexByte = hexByte - "0"
     ENDIF
     RETURN

WriteData:

     index = outByte >> 4
     READ index, highNibble

     index = outByte AND 0x0F
     READ index, lowNibble

     SERTXD (highNibble, lowNibble, CR,LF)

     RETURN
```