Column #77, September 2001 by Jon Williams:

Tool Time

I'm pretty darned sure that if comedian Tim Allen knew anything about microcontrollers he'd be a BASIC Stamp user. Of course, he'd rewire it for 220 volts and over-clock the PIC/SX to run at 10 gigahertz so he could point at it proudly and grunt like a pig (That's how I show off my Stamp projects, don't you?).

Tim and his f amous TV alter-ego both love tools – but then, don't we all? Tools are good. They help us build and create things. They're even better when they're free. And that's what we're going to chat about this month: a couple of freebie tools from Parallax. One is new, the other isn't but has some really neat surprises.

LCD REDUX

My favorite BASIC Stamp peripheral is the character LCD. I love them. I have a truckload on my desk and I use them all the time. I have 2x16, 2x20, 4x20; I have just about every flavor of Scott Edwards serial LCDs. I'll say it again: I love them.

LCDs are good because they're inexpensive (downright cheap in some cases), easy to connect to the Stamp and allow us to provide a lot of information to the outside world. Naturally I was thrilled when I learned that the BS2p would have direct support for parallel LCDs and that the commands would work very much like SEROUT and SERIN – allowing me to use the formatting modifiers. Does it get any better than this?

For LCD lovers like me – it just has. About the only thing tedious when using character LCDs is designing custom characters and adding the data to a PBASIC program. Until recently, I would do like you and create my characters on paper. Ugh! Too tedious.

Enter the LCD Character Creator. This nifty little program (okay, I wrote it, but I still think it's nifty) lets you design and save custom LCD characters and save the definitions to use later. It supports 5x7 and 5x10 character designs and even includes the standard character set (from the Hitachi HD44780A00 ROM) so you have a starting point. The on-screen preview graphic lets you see what your custom character will look like in a display next to other characters.

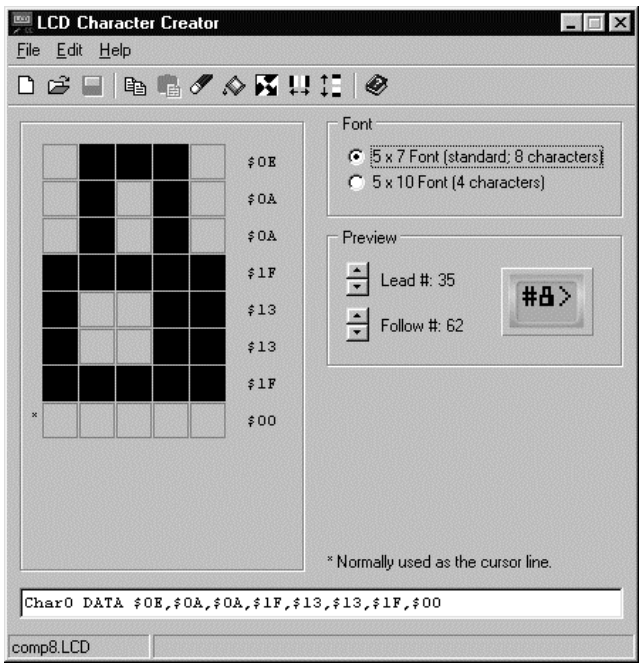
Essentially, this is a little paint program for LCD characters. Left-clicking on a pixel will toggle it on or off. There's also clear, fill, invert, mirror and flip commands. To make it easy to incorporate the new character into a PBASIC program, there's a text line at the bottom of the screen that can be copied right into your PBASIC source (this cool idea comes from Stamp user Steven M).

While the program is very easy to use, it comes with online help (Windows® HTML Help format) that explains all the menu items, includes a connection schematic (Figure 77.2 is right out of the help file) and demo programs for the BS2, BS2e, BS2sx and BS2p. It'll even point your browser to the Parallax web site.

COOL Digits

While playing one day I got the idea that I would create a custom set of digits for my LCD projects; you know, something that looked a little more sci-fi or computer-like. Creating the digits was no problem with my new program. But then – oops – I have ten digits and only eight custom characters in an LCD.

Figure 77.1: LCD Generator Character Map Program

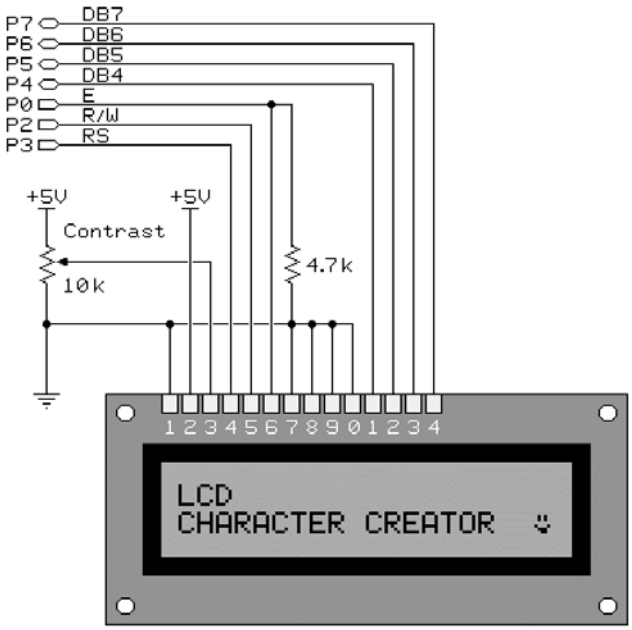


After scratching my head for a few minutes I remembered a trick that Scott Edwards taught me. If you change the definition of a custom character that is being displayed, the LCD will change along with it. So for this project, I changed my thinking from eight custom characters to eight spots on the LCD that I could update at any time. All I have to do is download new information to my assigned position.

Okay, let's do it. Take a look at Listing 77.1. This is a simple program – short and sweet. Its purpose is to display a running counter with my cool new digits. I'll use a Word variable to count from 0 to 999 and display it as 0.0 to 99.9 by inserting a decimal point between digits.

The EEPROM Data section contains the definitions for the new characters that were created with the LCD Character Creator program (you can download the files with the source code for this article). One note: The LCD Character Creator always names the data line "Char0. " After pasting each line into the source file it was renamed.

Figure 77.2: Schematic for BS2 to HD44780 LCD

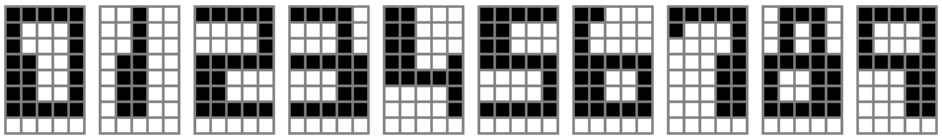


The name is used as a pointer to the start of the character. We need to know the location of each character in the EEPROM so we can download it when needed to the LCD.

The LCD is initialized to operate in 2-line mode. I used a 2x16 display for my experiment. I mention the width because it affects the placement of the right-justified numbers. Since we want to display three digits, we'll use custom character slots zero, one and two. Custom character zero will display tenths, digit one will display ones and digit two will display tens. Initialization is finished by downloading " new" zeros to digits zero and one, and a space to digit two.

Jump down to the `Update_CC`. This is the code that takes care of downloading an image map to the LCD. When we enter this routine we pass the custom character to update in `cNum` and the beginning EEPROM address of our new character data in `addr`.

Figure 77.3: High-Tech Digits



Since we've initialized the LCD to use two lines, we're forced to use the (standard) 5x7 character font which means eight bytes per character (the value of the `CLines` constant). The first line of `Update_CC` uses `LCDCMD` to point into the LCD's CGRAM (Character Generator RAM) based on the character we want to update and the number of bytes required for a definition. A `FOR-NEXT` loop grabs the new character from EEPROM with `READ`, then sends it to the LCD with `LCDOUT`. Simple and fast – which is exactly what we need since we're going to change characters on the fly.

Back in the main body of the program we spit out a label and put our characters onscreen where we want them. Remember that digit zero is tenths, so it will be right-most in the display.

The routine called `Show_Counter` is the heart of the program. The outer loop counts from zero to 999. The inner loop scans the value of counter using the `DIG` operator. This conveniently returns the digit value of the position scanned. It's a simple matter of using this value to point into our table of definitions for the map of that digit.

The only exception is when the tens digit is zero – we'd rather have a leading space than a leading zero. The code checks to see if the scanned digit is the tens position (character two) and its value is zero. When this is the case we download a custom space (all zeros). It's easier to re-define a custom space than to use the standard ASCII space character (#32). If we did this, we'd actually have more code to write. Eight bytes of EE space for a definition is an easy trade.

So that's it. When the program is run we'll see a counter with cool, high-tech looking digits (Figure 77.3).

Extending this idea, you could – with a bit of creativity – use it to build larger graphics. On a 2x16 display you could assign four characters to a 2x2 region. By downloading all four characters at once, you change the entire region, giving you quasi-graphic capabilities. It's certainly not as nice as one of the SEETRON graphical LCDs (you have character and line breaks), but it might get you by in a pinch.

An Updated Stamp Compiler

The latest release of the BASIC Stamp compiler is version 1.2. On the surface, this is a maintenance release, but under the hood...there's a couple of really neat new features.

The fixes include the DEBUG window problem that has been irritating WinNT users. I know they'll be happy to read that. The program also does a better job of identifying the attached Stamp and will even suggest the proper $Stamp directive. You can't download without a $Stamp directive now and a new menu and toolbar icons make it easy to insert and change.

For those of you that don't like looking for help in another document or in a book, there's a syntax help file. If you're having trouble with a keyword, highlight it and then press F1. Presto – you're in the help file and on the correct page. If you've misspelled the word, you're taken to an index page where you'll find the correct spelling (PULSOUT is very frequently misspelled PUSLEOUT). So the fixes and minor updates are nice, but let's talk about the really cool stuff....

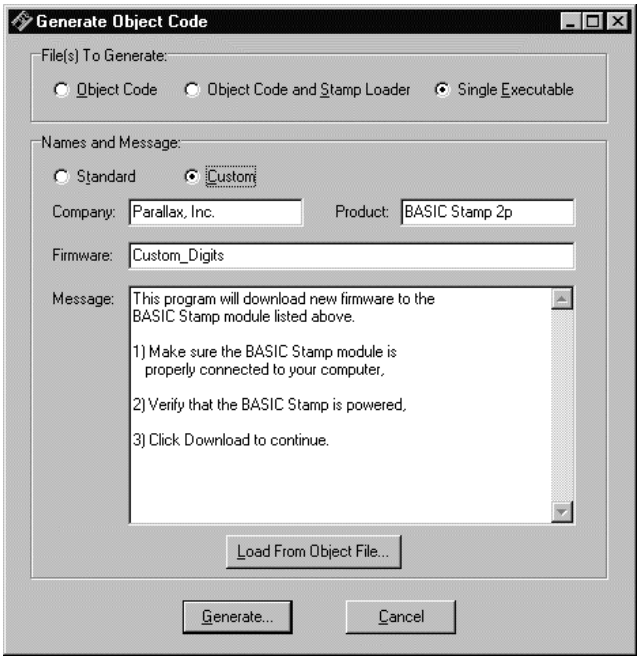
One of the longest running complaints about the BS2 compiler is the lack of BSAVE. For those of you who never programmed the BS1, BSAVE caused the compiler to generate an object file that was an image map of the Stamp EEPROM. This file could be downloaded to the Stamp through a small utility. What this allowed was the ability to update a Stamp 1 project without divulging source code.

The new Stamp compiler generates object code for any of the BS2 modules and it goes way beyond what the original BSAVE did. Figure 77.4 shows the dialog that results in clicking File > Generate Object Code. You'll see that there are three selections: Object Code, Object Code and Stamp Loader, and Single Executable.

The Object Code selection does just that, but the BS2 object file format goes beyond a simple EEPROM map. It also contains string information that is displayed in the Stamp Loader. By selecting Custom in the Names & Messages area we're able to customize the data displayed in the Stamp Loader.

The second selection, Object Code and Stamp Loader, is what you'll want to do the first time you run this process so that StampLoader.EXE is created. This program will load any BS2 object file and download it to a connected Stamp.

Figure 77.4: Stamp Editor Version 1.2 Object Code Generator

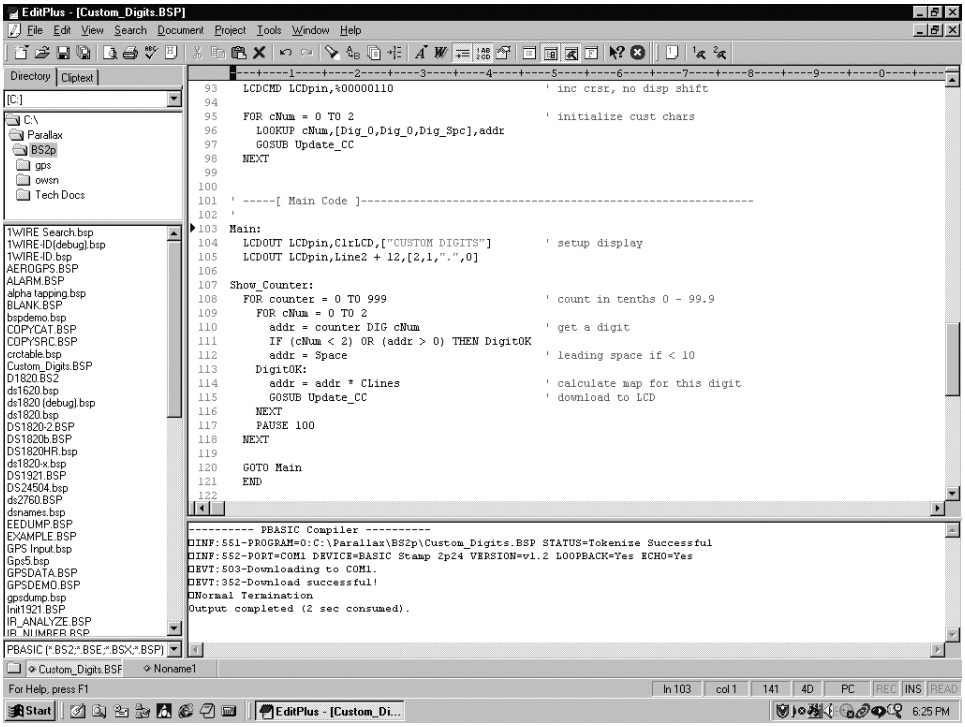


By doing this you can send StampLoader.EXE and several object files (perhaps variants on the same program) to a customer. StampLoader lets you select an object file and then downloads it at your command.

The final selections, Single Executable, is really exciting. By making this selection we can generate a small Windows program that will update a customer's Stamp-based project – and all the customer has to do is connect the project to a PC with a serial cable and run the program. This is a real winner for consultants. The look of the executable is customized by our settings in the Generate Object Code dialog (at the time of this writing I was working with a Beta version of the software – the final release will include even more customization).

The other major improvement to the Stamp editor is completely under the hood – you can't see it. The new compiler has a command-line interface that allows it to be controlled by other programs.

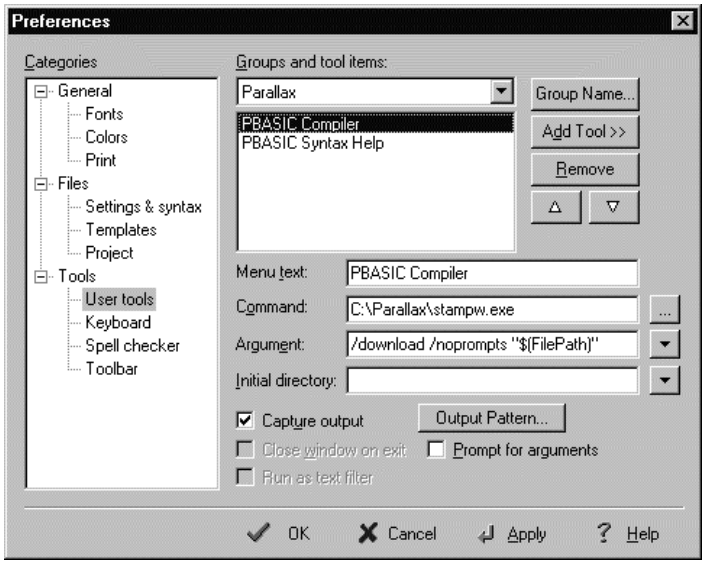
Figure 77.5: EditPlus Screen Shot with Stamp Code



What does this mean? For starters, it means that if your favorite programming editor can launch a program with command line parameters you can now use it to do Stamp development. You're no longer forced to write your Stamp code in the Stamp editor (but you do need to have the editor installed).

My favorite programming editor if EditPlus 2 (www.editplus.com). It's a $30 shareware program and consistently gets rave reviews. I like it because it has user-definable syntax highlighting and it does a great job with HTML files with its built-in browser. And since it was designed for programmers, EditPlus can launch other executables with command line parameters. It can even do keyword look-ups in your executable's help file.

Figure 77.6: EditPlus Setup To Program Stamps



Take a look at Figure 77.5. This is a screen shot of EditPlus after compiling and downloading the LCD program we just wrote. On the left edge is a file selection box, the main pane displays our syntax-highlighted source code and the bottom pane shows the (successful) output from the Stamp compiler.

My purpose here is not to teach you EditPlus, but to show how to connect your favorite programming editor to the Stamp compiler. It's important that the command line switches are correct. For EditPlus, I used this argument string:

/download /noprompts "$(FilePath)"

The first switch causes the PBASIC source file to be compiled and downloaded. If this process is not successful, you'll be alerted with an appropriate message. The second switch turns of normal message boxes – except DEBUG. I like this feature since I can still run programs that use DEBUG output from EditPlus. Of course, I can't switch between multiple DEBUG windows, but that's never been an issue for me so I'm not concerned.

The "$(FilePath)" parameter tells EditPlus to send the filename and complete path to the Stamp compiler. The reason it's quoted is that I use long filenames with spaces. Spaces in the filename will create a problem for the Stamp editor if you don't use the quotes. Better to be safe than sorry.

The Stamp compiler requires command-line controlled output to be directed to a standard output. With EditPlus (and other editors) this output can be captured so no file was specified. If your favorite editor doesn't capture this data, be sure to redirect the output to a text file. You can do that by adding this bit of text to your argument string:

> stampout.txt

If things don't proceed as you expect, you can open this file and find out why.

Well, that's about all we have time for this month. For those of you interested in EditPlus, I've included the PBASIC syntax definition in this month's project files. Be sure to check the Parallax web site for the availability of LCD Character Creator and the new Stamp compiler.

Couldn't Get Back Issues?

From time-to-time I'll get a question via e-mail that causes me to reference a past issue of Nuts & Volts. Between Scott Edwards, Lon Glazner and myself, we've written over 75 Stamp Applications articles. That's a lot of Stamp stuff. Unfortunately, back issues are not always available and occasionally left us scrambling to dig out old files.

Not any more. Ken Gracey of Parallax worked with Nuts & Volts to compile Stamp Applications columns 1 through 75 into a two-volume book set. Between the two books, there are nearly 1000 pages of BASIC Stamp programming tips and projects. All of the source code and related files are included on a CD.

You can order the books from Nuts & Volts or Parallax. Better do it quick though – these babies will go fast.

Until next time, Happy Stamping.

```
' Program Listing 77.1
' Nuts & Volts, September 2001

' -----[ Title ]-----------------------------------------------------------
'
' File..... Custom_Digits.BSP
' Purpose... Advanced LCD Demo - custom numeric font(s)
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' Started...
' Updated...

' { $STAMP BS2p }


' -----[ Program Description ]---------------------------------------------
'
' This program demonstrates character definition replacement in order to
' create a custom font for numbers.  This program uses three of the eight
' custom (CGRAM) character slots to display the tens, ones and tenths value
' of a counter.
'
' The program analyzes the counter and updates the screen by downloading
' the appropriate character map for each digit.
'
' To run this program on the BS2p Demo Board, connect the LCD and
' install Jumper X3. Adjust contrast pot for best display.
'
' Refer to the Hitachi HD44780 documentation for details on LCD control.


' -----[ Revision History ]------------------------------------------------
'


' -----[ I/O Definitions ]-------------------------------------------------
'
LCDpin   CON     0                      ' connect LCD to OutL


' -----[ Constants ]-------------------------------------------------------
'
NoCmd    CON     $00                    ' No command in LCDOUT
ClrLCD   CON     $01                    ' clear the LCD
CrsrHm   CON     $02                    ' move cursor to home position
CrsrLf   CON     $10                    ' move cursor left
CrsrRt   CON     $14                    ' move cursor right
DispLf   CON     $18                    ' shift displayed chars left
DispRt   CON     $1C                    ' shift displayed chars right
DDRam    CON     $80                    ' Display Data RAM control
CGRam    CON     $40                    ' Custom character RAM
Line1    CON     $80                    ' DDRAM address of line 1
Line2    CON     $C0                    ' DDRAM address of line 2

CLines   CON     8                      ' lines per character
Space    CON     10
```

```
' -----[ Variables ]------------------------------------------------------------
'
char             VAR     Byte             ' character sent to LCD
addr             VAR     Byte             ' EE starting address of map
cNum             VAR     Nib              ' character number
idx              VAR     Nib              ' loop counter
counter          VAR     Word


' -----[ EEPROM Data ]-----------------------------------------------------------
'
' character definitions - digits 0 - 9 and space
'
Dig_0            DATA    $1F,$11,$11,$19,$19,$19,$1F,$00
Dig_1            DATA    $04,$04,$04,$0C,$0C,$0C,$0C,$00
Dig_2            DATA    $1F,$01,$01,$1F,$18,$18,$1F,$00
Dig 3            DATA    $1E,$02,$02,$1F,$03,$03,$1F,$00
Dig 4            DATA    $18,$18,$18,$19,$1F,$01,$01,$00
Dig 5            DATA    $1F,$18,$18,$1F,$01,$01,$1F,$00
Dig_6            DATA    $18,$10,$10,$1F,$19,$19,$1F,$00
Dig_7            DATA    $1F,$11,$01,$03,$03,$03,$03,$00
Dig_8            DATA    $0E,$0A,$0A,$1F,$13,$13,$1F,$00
Dig_9            DATA    $1F,$11,$11,$1F,$03,$03,$03,$00
Dig Spc          DATA    $00,$00,$00,$00,$00,$00,$00,$00


' -----[ Initialization ]--------------------------------------------------------
'
Initialize:
  PAUSE 500                              ' let the LCD settle
  LCDCMD LCDpin,%00110000 : PAUSE 5      ' 8-bit mode
  LCDCMD LCDpin,%00110000 : PAUSE 0
  LCDCMD LCDpin,%00110000 : PAUSE 0
  LCDCMD LCDpin,%00100000 : PAUSE 0      ' 4-bit mode
  LCDCMD LCDpin,%00101000 : PAUSE 0      ' 2-line mode
  LCDCMD LCDpin,%00001100 : PAUSE 0      ' no crsr, no blink
  LCDCMD LCDpin,%00000110                ' inc crsr, no disp shift

  FOR cNum = 0 TO 2                      ' initialize cust chars
    LOOKUP cNum,[Dig_0,Dig_0,Dig_Spc],addr
    GOSUB Update_CC
  NEXT


' -----[ Main Code ]-------------------------------------------------------------
'
Main:
  LCDOUT LCDpin,ClrLCD,["CUSTOM DIGITS"]   ' setup display
  LCDOUT LCDpin,Line2 + 12,[2,1,".",0]

Show_Counter:
  FOR counter = 0 TO 999                 ' count in tenths 0 - 99.9
    FOR cNum = 0 TO 2
      addr = counter DIG cNum            ' get a digit
      IF (cNum < 2) OR (addr > 0) THEN DigitOK
      addr = Space                       ' leading space if < 10
    DigitOK:
      addr = addr * CLines               ' calculate map for this digit
```

```
      GOSUB Update_CC                       ' download to LCD
    NEXT
    PAUSE 100
  NEXT

  GOTO Main
  END


' -----[ Subroutines ]------------------------------------------------------
'
Update CC:                                  ' update custom character
  LCDCMD LCDpin,(CGRam + (cNum * CLines))   ' point to start of character map
  FOR idx = 0 TO (CLines - 1)
    READ (addr + idx),char                  ' get data for character line
    LCDOUT LCDpin,NoCmd,[char]              ' write to LCD CGRAM
  NEXT
  RETURN
```