Stamp Applications no. 17 (July '96):

Exploring the BS1 EEPROM With a Homemade Browser

Viewing and Modifying Long-Term Data

by Scott Edwards

IT AIN'T MUCH, that 256 bytes of storage, but it's arguably the most important aspect of the BASIC Stamp I (BS1) or its kit brother, the Counterfeit. So this month's column will focus on program memory. After a little background, I'll present techniques for viewing and modifying data stored in memory using a free QBASIC program on your PC.

A place for your programs. When you program a BS1 or Counterfeit, the host software on your PC converts your BASIC program into *tokens*, digital shorthand that the interpreter chip understands. It sends the tokens through the downloading cable to the Stamp, which writes them into its EEPROM (electrically erasable, programmable, read-only memory). The neat thing about EEPROMs is that they retain their contents with the power turned off, but can be reprogrammed over and over. Those virtues make EEPROMs a great place to store programs and long-term data.

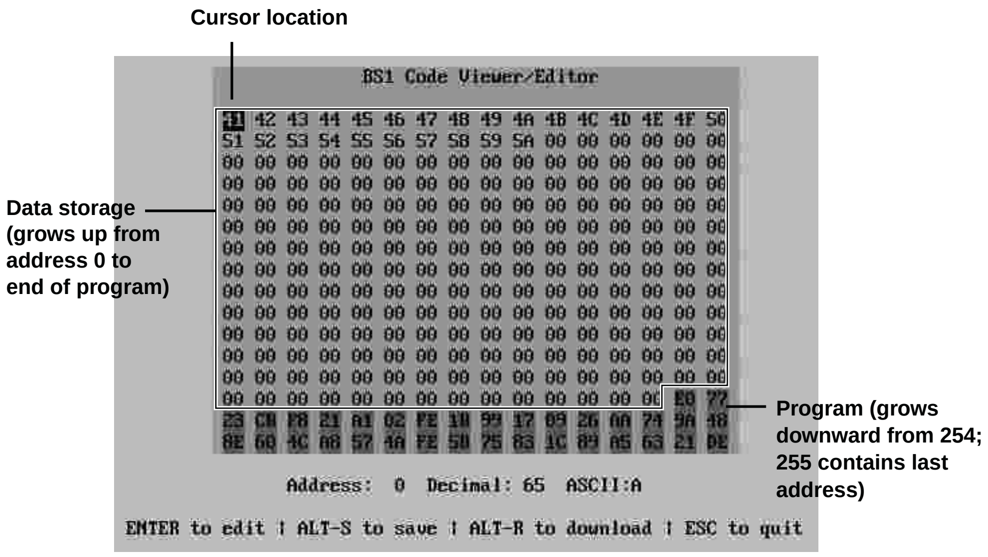
EEPROMs eventually wear out after many programming cycles. The devices used in the Stamps are currently specified to last at least 1 million programming cycles. (Some parts of the documentation still say 100,000 programming cycles, which was true at the time they were written. Improved manufacturing processes continue to increase programming *endurance*, so I wouldn't be surprised to find out that they're now rated at 2 million cycles or more.)

The particular EEPROM used in the BS1 and Counterfeits is the 93LC56. It holds 256 bytes of data, enough for a BASIC program 70+ instructions long, or a mixture of instructions and data.

There are two ways to get data into the BS1's EEPROM: (1) You can tell the host program, STAMP.EXE, to store the data at the same time it is downloading the program by using the EEPROM directive; or (2) You can have your program store data into the EEPROM using the WRITE instruction.

A quick note about the language of the previous paragraph: I called EEPROM a *directive* because it tells the host software what to do (put bytes into the EEPROM), but does not change the actual PBASIC program. WRITE is an *instruction* because it does become part of the program. Directives work at *compile time*, when the host computer is setting up to download your program. Instructions work at *run time*, when the program is actually running.

There are also two ways to get data back out of the EEPROM: (1) You can use the BSAVE directive to tell STAMP.EXE to create a file on your PC's disk drive containing a copy of all 256 bytes as it is preparing to download them to the EEPROM; or (2) You can use the instruction READ in your program to examine any EEPROM byte.


Cursor location
Data storage (grows up from address 0 to end of program)
Program (grows downward from 254; 255 contains last address)

```
                  BS1 Code Viewer/Editor

41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50
51 52 53 54 55 56 57 58 59 5A 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 E0 77
23 C8 F8 21 A1 02 FE 18 99 17 09 26 AA 74 9A 48
8E 60 4C A8 57 4A FE 58 75 83 1C 89 A5 63 21 DE

          Address:  0  Decimal: 65  ASCII:A

ENTER to edit | ALT-S to save | ALT-R to download | ESC to quit
```

Figure 1. Screen shot of BS1 object-code browser graphically illustrates how the Stamp stores data and programs in EEPROM.

The Stamp programming disk includes a utility called BSLOAD that takes the file created by BSAVE (called an *object* file) and downloads it to a Stamp connected to the PC. It's most often used to keep a PBASIC program private; someone can download a BSAVEd program to a Stamp without seeing (or messing with) the actual BASIC instructions. Application note no. 16 available from Parallax (and included with the Counterfeit) gives a bit more background on BSAVE.

Beyond BSAVE. It's all well and good to distribute program updates as PBASIC object code and allow a user to download them with BSAVE, but how about giving the user a limited ability to customize the operation of the Stamp-based device? If your PBASIC program relies on values stored in EEPROM to determine its behavior (like the rotating message display from April's column) it's a piece of cake.

The trick is to understand how the BSAVE object file is structured, and how to read, modify, and save it using familiar PC tools. I'll demonstrate using QBASIC, the free and surprisingly modern dialect of BASIC that Microsoft has given away with every version of DOS since 5.0.

The program, called BS_PEEK.BAS, is available from the Parallax BBS or FTP site (Sources). Figure 1 shows a sample screen from the program; listing 1 is the program itself.

• *Loading the File.* PBASIC object code files consist of 256 bytes of data arranged backwards with respect to the BS1's READ/WRITE addressing system. In other words, the address that you call 255 for the sake of a READ or WRITE is actually the very first byte contained in an object file. In order to preserve the BS1's addressing scheme, our QBASIC program for the PC will load the file backwards, as shown in this subroutine:

```
SUB readFile
OPEN filename FOR INPUT AS #1
FOR i = 255 TO 0 STEP -1
    code(i) = ASC(INPUT$(1, 1))
NEXT i
CLOSE #1
END SUB
```

Those instructions load the file specified by the string "filename" into a 256-byte array called "code(n)". You can readily see the backwards part: the file is read one byte at a time from start to end, but it's stored into the code(n) array starting at index 255 and working down to 0.

You may remember from the PBASIC manual entry on the WRITE instruction that program memory starts at address 254 and works downward, depending on the length of the program. Address 255 contains the lowest address used by the program; everything below that may be used to store long-term data. By long-term, I mean data that doesn't go away when the power is disconnected.

Our browser program uses the value stored in address 255 to display the program memory in a different color (red) and to prevent users from changing it. The logic is the same as for a PBASIC program:

```
IF cursPos >= code(255) THEN BEEP
```

In other words, if the user has the cursor on one of the program memory bytes and wants to change the value of that byte, the program beeps and refuses to do it. Since PBASIC is stored in variable-length tokens whose coding is not documented, no good could come of letting users mess with program memory.

• *Modifying the File.* Once the code is in memory, modifying it is just a matter of changing the content of the code(n) array. Write a value to index 0 of that array, and it'll later be written to address 0 of the Stamp's EEPROM. Simple as that.

• *Saving the File.* Since we loaded the file in reverse order, we've got to reverse it again when we save it:

```
SUB saveFile
OPEN filename FOR OUTPUT AS #1
FOR i = 255 TO 0 STEP -1
  PRINT #1, CHR$(code(i));
NEXT i
CLOSE #1
END SUB
```

The subroutine that does this is almost identical to the readFile subroutine.

• *Downloading.* Rather than ask the user to run BSLOAD on the newly saved file, I thought it would be neat to permit downloading from within the browser program. QBASIC offers a method for executing DOS commands or other programs called SHELL. The jargon is that you "shell out to DOS" to take care of some business, then return when done. All you have to do is execute the instruction SHELL, followed by a text string containing the stuff you would have typed at the DOS prompt:

```
CLS: theShell$="BSLOAD "+filename
SHELL theShell$
```

The first line above clears the screen and creates the string that SHELL is to execute; the second line carries out the instruction.

If you've ever used BSLOAD, you know that it presents a few lines of text indicating that a download is in progress, then prints a message to indicate whether or not it was successful using the terms "OK" or "ERROR." It does not tell the user what to do in the event of an ERROR. Since BSLOAD is not really designed for use by other programs, it doesn't use the DOS error-reporting method, so QBASIC isn't informed of the problem, either.

For the program to be useful by non-technical people, it should be aware of a problem with the download and suggest solutions (check cable, battery, etc.). The program gets this information the same way you would: by reading the screen. It loads all of the text from the 80-column by 25-line DOS screen into an array, then searches that array for the keyword "ERROR." This is a sufficiently useful trick for QBASIC that I wrote it as a function that can search the screen for any string and return an integer representing that string's position on the screen. If the string isn't there, the function returns zero. See the function screenSays in listing 1. Once this function is defined, checking for the word "ERROR" in BSLOAD's screen output goes like this:

```
IF screenSays%("ERROR") THEN...
```

with the consequences of the error following THEN.

Wrapup. Now you know how to manipulate the

BS1 EEPROM contents. The program I've presented almost certainly doesn't do the particular job that you need done, but it illustrates every important, Stamp-specific function. Please don't bombard me with requests for custom features, or translations to Visual BASIC or C—writing your program is your job!

A couple of final notes: I asked Parallax about some of the drawbacks of the existing BSLOAD utility, such as the lack of error reporting, and the verbose screen message that displays the BASIC Stamp name and Parallax' phone number. They said that they're willing to spruce up BSLOAD to meet the requirements of manufacturers or others who need to make heavy-duty use of this capability.

I also asked about a BSAVE/BSLOAD capability for the Stamp II (BS2). There isn't one right now, but it may be incorporated into the upcoming BS2+ design.

Where's BASIC for Beginners? That department is taking a break this month. Its allotted time was snarfed up by negotiations to publish a beginners' PBASIC *book*. Details and schedule are currently as vague as election-year platforms, but it looks like a good bet.

NOTE: This article was originally published in 1996. The Stamp Applications column continues with a changing roster of writers. See www.nutsvolts.com or www.parallaxinc.com for current Stamp-oriented information.

Listing 1. QBASIC Browser for BS1 Object Code

```
' QBASIC (PC) Program: BS_PEEK.BAS
' This program allows you to view, modify, save, and download Stamp I
' (BS1) object-code files created by the BSAVE directive. It provides
' examples of techniques for reading and writing BS1 object files.
' To use the downloading capability, you must have the BSLOAD utility
' on your computer and either in the current directory, or in your
' PATH statement. This program is not meant to be the ultimate in
' object-code editing--just a suitable starting point for more elaborate
' programs.

'======================SUBROUTINES=======================
DECLARE FUNCTION screenSays% (searchString AS STRING)
DECLARE SUB acceptEntry ()
DECLARE SUB goHome ()
DECLARE SUB goEnd ()
DECLARE SUB saveFile ()
DECLARE SUB rightKey ()
DECLARE SUB upKey ()
DECLARE SUB downKey ()
DECLARE SUB leftKey ()
DECLARE SUB makeBkgd ()
DECLARE SUB readFile ()
DECLARE SUB updateCell (cell AS INTEGER)
DECLARE FUNCTION HEXX$ (NUM AS INTEGER)
DEFINT A-Z
'======================CONSTANTS=======================
CONST black = 0:   CONST blue = 1
CONST green = 2:   CONST cyan = 3
CONST red = 4:     CONST magenta = 5
CONST brown = 6:   CONST white = 7
CONST foreground = white
CONST background = blue
CONST lockout = red
enter$ = CHR$(13)
'======================SCAN CODES=======================
leftArrow$ = CHR$(0) + CHR$(75)
rightArrow$ = CHR$(0) + CHR$(77)
upArrow$ = CHR$(0) + CHR$(72)
downArrow$ = CHR$(0) + CHR$(80)
home$ = CHR$(0) + CHR$(71)
end$ = CHR$(0) + CHR$(79)
altS$ = CHR$(0) + CHR$(31)
altR$ = CHR$(0) + CHR$(19)
esc$ = CHR$(27)
'======================GLOBAL VARIABLES=======================
DIM SHARED code(255) AS INTEGER
DIM SHARED attribs(255) AS INTEGER
DIM SHARED cursPos AS INTEGER
DIM SHARED filename AS STRING

'======================MAIN PROGRAM=======================
' Load the file specified by COMMAND$ (QuickBASIC compiler only)
' or get the user to enter a file path.
start:
CLS
LOCATE 5, 1: INPUT ; "Enter pathname of BS1 object file: ", filename
IF filename = "" THEN GOTO start
continue:
CLS
makeBkgd
readFile
' Assign attributes to each screen cell (byte of data) depending on
' whether it's program or data. The last program-memory address is
' stored in location 255.
FOR i = 0 TO 255
  IF i < code(255) THEN
    attribs(i) = 0
  ELSE
    attribs(i) = 2
  END IF
```

```
NEXT
' Display the data on the screen. Place the cursor at 0.
FOR i = 0 TO 255
 updateCell (i)
NEXT
cursPos = 0: attribs(cursPos) = 1
updateCell (cursPos)
' Scan for keys pressed by the user and respond accordingly.
scanit:
k$ = INKEY$
IF k$ = "" THEN GOTO scanit
SELECT CASE k$
   CASE leftArrow$
     leftKey
   CASE rightArrow$
     rightKey
   CASE upArrow$
     upKey
   CASE downArrow$
     downKey
   CASE home$
     goHome
   CASE end$
     goEnd
   CASE enter$
     acceptEntry
   CASE altS$
     saveFile
   CASE altR$
     CLS : theShell$ = "BSLOAD " + filename
     SHELL theShell$
     IF screenSays%("ERROR") THEN
       CLS : PRINT "Download failed!": PRINT
       PRINT "Check cable for correct connection. "
       PRINT "Ensure that power supply is connected."
       INPUT ; "Press ENTER to continue.", dummy$
       GOTO continue
     ELSE
       CLS : PRINT "Download successful."
       INPUT ; "Press ENTER to continue.", dummy$
       GOTO continue
     END IF
   CASE esc$
     END
END SELECT
GOTO scanit
```

```
SUB acceptEntry
' Let the user enter a new decimal value for cell that the
' cursor is now on. If it's in program memory, beep and
' ignore the entry.
IF cursPos >= code(255) THEN
  BEEP: EXIT SUB
ELSE
  LOCATE 22, 22: INPUT ; "new value (decimal): ", newVaL$
  code(cursPos) = (VAL(newVaL$) AND 255)
  updateCell (cursPos)
  LOCATE 22, 22: PRINT SPC(58);
END IF
END SUB

SUB downKey
' Respond to down-arrow key by moving the cursor to the next row.
attribs(cursPos) = attribs(cursPos) - 1
updateCell (cursPos)
cursPos = (cursPos + 16) AND 255
   attribs(cursPos) = attribs(cursPos) + 1
updateCell (cursPos)
END SUB

SUB goEnd
' Respond to the END key by going to the last location
' in data memory.
attribs(cursPos) = attribs(cursPos) - 1
updateCell (cursPos)
cursPos = (code(255) - 1) AND 255
   attribs(cursPos) = attribs(cursPos) + 1
updateCell (cursPos)
END SUB

SUB goHome
' Respond to the HOME key by going to the 0th location.
attribs(cursPos) = attribs(cursPos) - 1
updateCell (cursPos)
cursPos = 0
   attribs(cursPos) = attribs(cursPos) + 1
updateCell (cursPos)
END SUB

FUNCTION HEXX$ (NUM AS INTEGER)
' Convert 8-bit integer to 2-digit hex values with leading 0s.
 IF NUM > 15 THEN
   HEXX$ = HEX$(NUM)
 ELSE
   HEXX$ = "0" + HEX$(NUM)
 END IF
END FUNCTION
```

```
DEFINT A-Z
SUB leftKey
' Respond to left arrow by moving back one cell.
attribs(cursPos) = attribs(cursPos) - 1
updateCell (cursPos)
cursPos = (cursPos - 1) AND 255
   attribs(cursPos) = attribs(cursPos) + 1
updateCell (cursPos)
END SUB

SUB makeBkgd
' Print instructions. Draw a color rectangle on the screen and add
' program title to it.
LOCATE 21, 22: PRINT "Address:     Decimal:     ASCII: "
LOCATE 23, 7
PRINT "ENTER to edit | ALT-S to save | ALT-R to download | ESC to quit";
FOR i = 2 TO 19
  COLOR foreground, background
  LOCATE i, 15
  PRINT SPC(49);
NEXT
  LOCATE 2, 29: PRINT "BS1 Code Viewer/Editor"
END SUB

SUB readFile
' Read in the 256-byte BS1 object code file and store
' it in the code(n) array.
OPEN filename FOR INPUT AS #1
FOR i = 255 TO 0 STEP -1
        code(i) = ASC(INPUT$(1, 1))
NEXT i
CLOSE #1
END SUB

SUB rightKey
' Respond to right arrow key by moving forward one data cell.
attribs(cursPos) = attribs(cursPos) - 1
updateCell (cursPos)
cursPos = (cursPos + 1) AND 255
   attribs(cursPos) = attribs(cursPos) + 1
updateCell (cursPos)
END SUB

SUB saveFile
' Save the code array to current filename.
LOCATE 22, 22: PRINT "saving: "; filename;
OPEN filename FOR OUTPUT AS #1
FOR i = 255 TO 0 STEP -1
  PRINT #1, CHR$(code(i));
NEXT i
```

```
CLOSE #1
LOCATE 22, 22: PRINT SPC(58);
END SUB

FUNCTION screenSays% (searchString AS STRING)
' Load the current contents of the 25x80 DOS screen into a
' string and search that string for "searchString." Return
' an integer representing the position of the string on the
' screen, or 0 if the string isn't found.
screenText$ = ""
FOR i% = 1 TO 25
  FOR j% = 1 TO 80
    screenText$ = screenText$ + CHR$(SCREEN(i%, j%))
  NEXT
NEXT
screenSays% = INSTR(screenText$, searchString)
END FUNCTION

SUB updateCell (cell AS INTEGER)
' Print the specified data cell to the screen.
' Use its attributes to determine color.
cellRow = 4 + (cell \ 16)
cellCol = 16 + (3 * (cell MOD 16))
LOCATE cellRow, cellCol
SELECT CASE attribs(cell)
  CASE 0
    COLOR foreground, background
  CASE 1
    COLOR background, foreground
  CASE 2
    COLOR foreground, lockout
  CASE 3
    COLOR lockout, foreground
END SELECT
PRINT HEXX$(code(cell));
LOCATE 21, 30: COLOR white, black: PRINT USING "###"; cell;
LOCATE 21, 43: PRINT USING "###"; code(cell);
LOCATE 21, 54: PRINT " ";
LOCATE 21, 54: PRINT CHR$(code(cell));
END SUB

SUB upKey
' Respond to the up arrow by moving up a row.
attribs(cursPos) = attribs(cursPos) - 1
updateCell (cursPos)
cursPos = (cursPos - 16) AND 255
   attribs(cursPos) = attribs(cursPos) + 1
updateCell (cursPos)
END SUB
```