THE APPLE SCREEN — II

Building A Subroutine Library

by Don Ravey
127 Chukker Court
San Mateo, CA 94403

As promised in Part I, we will now examine some specific subroutines that you might like to have as Text Files, ready to be EXEC'd into your Applesoft programs. We hope that you have established a **CAPT SUBR** program, as suggested in Part I, to make it easy to create your Text Files. In case you haven't, you may wish to get the last issue of NIBBLE and carefully read Part I of this series.

(Y/N)?

The first subroutine to be described is one I call **(Y/N)?**. It prints a string, ST$ (ALL my subroutine input strings are named ST$—not very imaginative perhaps, but consistent), followed by **(Y/N)?**. It then waits for a keystroke answer. If the **Y** key is pressed, it sets the variable **YES** to **TRUE (1)**. Any other key will set **YES** to **FALSE (0)**. Your next program statement can then be something like **IF YES THEN . . .**, making your program very easy to read! I use line number 102 for this simple routine:

```
102 PRINT ST$; "(Y/N)?";: GET A$:
    PRINT A$: YES = A$ = "Y": RETURN
```

That's not a misprint just before the "Return" — that's a boolean expression! Would it help if I wrote it: YES = (A$ = "Y")? The expression inside the parentheses is either TRUE (1) or FALSE (0); therefore, **the variable YES is assigned a value of 1 or 0**.

Let's look at how you might use this in an Applesoft program:

```
 10 REM ** SAMPLE PROGRAM **
 20 TEXT : HOME : REM ALWAYS A
    GOOD WAY TO START A PROGRAM
 30 D$ = CHR$(4)
 99 GOTO 199 : REM SKIP OVER
    SUBROUTINES
102 PRINT ST$; " (Y/N)?";: GET A$:
    PRINT A$: YES = A$ = "Y": RETURN
199 REM BEGIN MAIN PROGRAM:
200 ST$ = "DO YOU LIKE THIS
    SUBROUTINE": VTAB 10: GOSUB 102
210 IF YES THEN 250
220 PRINT "TOO BAD!"
230 GOTO 300
250 PRINT "GLAD YOU LIKED IT."
300 REM CONTINUE
```

To use this subroutine, note that you assign the desired string to ST$ and determine your screen position before calling the subroutine (see line 200).

SLOWPRINT

On occasion, I like to have one or more lines "spit out" the characters, one after another, instead of the whole line or paragraph going "splat" on the screen at once.

The routine is very simple. If you set **SPEED = 200** or so, just before calling the subroutine, the action will be slower (don't forget to later reset SPEED = 255!). I put the "plain" routine on line number **104** and a "sound effects" version on line **105**. The expression: **Z = PEEK (-16336) + PEEK (-16336)** is just a way of referencing the loudspeaker output location twice — the "Z" value isn't used for any purpose. We use two references because each reference to that location "toggles" (alternates) the state of the speaker, and the "click" that we want to produce occurs on only one of these transitions.

To slow down the action a little, I've added a "do nothing" loop inside the "PRINT" loop.

```
104 FOR X = 1 TO LEN (ST$): PRINT MID$
    (ST$, X, 1);: FOR Y = 1 TO 20: NEXT:
    NEXT: PRINT: RETURN
105 FOR X = 1 TO LEN (ST$): PRINT MID$
    (ST$, X, 1);: Z = PEEK (-16336) + PEEK
    (-16336): FOR Y = 1 TO 20: NEXT:
    NEXT: PRINT: RETURN
```

SLOWERASE

Now that we have "slow-printed" a line to the screen, one letter at a time (with the line 105 "sound effects" version, it sounds quite like a typewriter), what about erasing it a letter at a time, from the end back to the beginning of the line?

It turns out that Applesoft makes this very easy to do, and the visual effect is quite attractive. The key to doing this is the Applesoft subroutine that starts at memory address **-868** (that's the same as **65,536 - 868 = 64,668**, but isn't -868 easier to remember?!). The **CALL -868** routine is the same "clear to end-of-line" function that you get in immediate mode by pressing **control-E** (holding down "CTRL" while you press "E").

So all we have to do is set up a loop that progressively HTABs to lower and lower values **(STEP -1)**, starting with the end of the string, and clears everything past that position on the line. Oh, yes: we should return the cursor to the same line as it was when we started; it is a good programming practice to arrange for each subroutine to leave things as they were prior to calling the subroutine. This practice will pay off in easier debugging of complicated programs; it's a good idea to establish good habits early!

Memory address **37 (decimal)** is where the Apple stores the current cursor vertical position (often referred to as **CV**). Due to the sequencing of the machine language I/O routines, the direct use of the cursor position value can become a little tricky. I found, after a little experimenting, that in this case it's necessary to VTAB to the CV value before starting the "erase" loop, then reset CV to one less than its value, at the end of the loop, in order to tidy everything up.

For a "sound" version (line 107), just add the references to the speaker toggle address **(-16336)** in the loop, as in the "Slowprint" routine.

```
106 VTAB (PEEK (37)): FOR X = LEN (ST$)
    TO 1 STEP -1: HTAB X: CALL -868:
    FOR Y = 1 TO 40: NEXT: NEXT:
    POKE 37, PEEK (37) - 1: RETURN
107 VTAB (PEEK (37)): FOR X = LEN (ST$)
    TO 1 STEP - 1: HTAB X: CALL -868:
    FOR Y = 1 TO 40: NEXT: Z = PEEK
    (-16336) + PEEK (-16336): NEXT:
    POKE 37, PEEK (37) - 1: RETURN
```

BACKERASE

But, what if we want to erase a line from left to right? It will take a different technique, since CALL -868 clears to the end of the line. Aha! Where IS the end of the line? It's wherever we say it is, in memory address **33 (decimal)**! The strategy for this one is to generate a loop that repeatedly increases the defined text window width, then clears the line, as defined. By this time, you will no doubt see how to construct the loops, and the "sound" version:

```
108 FOR X = 2 TO 40: POKE 33, X:
    CALL -868: FOR Y = 1 TO 30: NEXT:
    NEXT: POKE 37, PEEK (37) - 1:
    RETURN
109 FOR X = 2 TO 40: POKE 33, X:
    CALL -868: FOR Y = 1 TO 30: NEXT:
    Z = PEEK (-16336) + PEEK (-16336):
    NEXT: POKE 37, PEEK (37) - 1:
    RETURN
```

CENTERPRINT

This one is as straightfoward as it is useful. It simply prints enough spaces before the string to center the string on the screen. How many spaces? Half the difference between the string length and 40 (the number of columns on the screen).

```
110 ST% = (40 - LEN (ST$)) / 2:
    PRINT SPC (ST%); ST$: RETURN
```

THE SLIDER

"CENTERPRINT" makes neat displays, but lacks action, or "Pizzazz"! Here's a variation that "slides" your ST$ string onto the screen from the right edge, leaving it centered. It's very eye-catching. For this we perform a loop that HTABs to lower and lower values, and prints as much of the string as will fit on the screen each time. If you're not comfortable with the LEFT$ Applesoft function, now is an excellent time to get acquainted with it, in your Applesoft Reference Manual — you'll find the Applesoft string handling functions extremely useful.

Each trip through the loop prints over what was printed by the previous loop. Note that each PRINT adds a **space (" ")** to the end of the string, to erase the last character of the previous PRINT, after the whole string fits on the screen. Another thing to watch for: don't PRINT too much of the string each time, or it will spill over to the next line. Again I found it necessary to experiment a little with the vertical tabbing, so that all the action occurs on the right line.

```
112 L = PEEK (37): FOR X = 1 TO 19 + LEN
    (ST$) / 2: VTAB L + 1: HTAB 40 - X:
    PRINT LEFT$ (ST$, X); "  ";: NEXT:
    PRINT: RETURN
```

BILLBOARD

If you have a l-o-n-g message instead of a few words, here's a "billboard" or "Times Square" routine that will handle over four lines of text — over **170** characters. By adding 40 spaces at the beginning and end of the ST$ string, the smooth entry and exit of your message is made MUCH simpler: we simply PRINT a 40-character MID$ string, advancing the starting point each time through. If you REALLY MUST display a longer message, you could rewrite the routine to do special things at the beginning and end, and utilize the maximum string length of 255 characters — but why make things difficult?

```
113 L = PEEK (37): SU$ =
    "          (40 spaces)          ":
    ST$ = SU$ + ST$ + SU$: FOR X = 1 TO
    LEN (ST$) - 40: VTAB L: PRINT MID$
    (ST$, X, 40): NEXT: PRINT: RETURN
```

SCREEN WIPES

We saw how the "erase" lines from one or the other end, by using the "clear to end-of-line" from within a loop. Now, let's apply similar techniques to "erase" the whole screen in various directions. You may have seen these routines in an earlier NIBBLE issue, identified as "Humble" (Be It Ever So Humble, "HOME" Is Noplace!).

For a **left-to-right** "Wipe", just keep defining the left edge of the text window further and further left, starting near the right edge —but you must also define the line width so that it won't exceed 40 — then "HOME":

```
120 FOR X = 38 TO 0 STEP -1: POKE 32, X:
    POKE 33, 40 - X: HOME: NEXT:
    RETURN
```

For a **right-to-left** "Wipe", it's even simpler — define the text window longer and longer, from the left edge, and do your "HOME":

```
121 FOR X = 2 TO 40: POKE 33, X: HOME:
    NEXT: RETURN
```

A **top-to-bottom** "Wipe"? What could be easier! Set the bottom of the text window progressively lower:

```
122 FOR X = 1 TO 24: POKE 35, X: HOME:
    NEXT: RETURN
```

I probably don't even have to tell you how to do a **bottom-to-top**:

```
123 FOR X = 23 TO 0 STEP -1: POKE 34, X:
    HOME: NEXT: RETURN
```

Next month, we will provide some very useful formatting subroutines, along with more complicated "novelty" routines.