Stamp Applications no. 3 (May '95):

Adapt a Keypad for Pro-Quality Data Entry; Included Software Lets Stamp "Type" on a PC

Hacking a Commercial Keypad, by Scott Edwards

NOTE: This article was originally published in 1995. The keypad described here is no longer available. Unless you have or find one of these keypads, this information is essentially obsolete. Sorry! Time marches on.

THIS month's applications make an off-the-shelf numeric keypad for PCs do double duty; first as a serial data-entry terminal for the Stamp, then as a sneaky software method of getting Stamp data into your favorite PC applications, like spreadsheets and word processors.

I've wanted to manufacture a Stamp-compatible keypad, but the cost of starting from scratch is daunting. A decent keypad requires good switches and a sturdy enclosure, both of which are very expensive in the small quantities that the Stamp aftermarket might use. There are occasional surplus bargains, but there's also the risk that the supply will dry up on the very day that Ultra-Mega Industries Inc. places a big order.

I recently found an accessory keypad for laptop computers—an ORTEK MCK-18S/N, made in Taiwan. It communicates with and is powered by a PC's serial port. Included software redirects data from the keypad to the keyboard buffer, so that it functions just like keys on the standard keyboard. The pad has 18 buttons, consisting of the numbers 0 through 9, decimal point, Escape, Num Lock, Enter, and the math operators (/, *, -, +). The keys are quality Alps switches with standard PC-style keycaps. The unit is enclosed in a nice flat-black case.

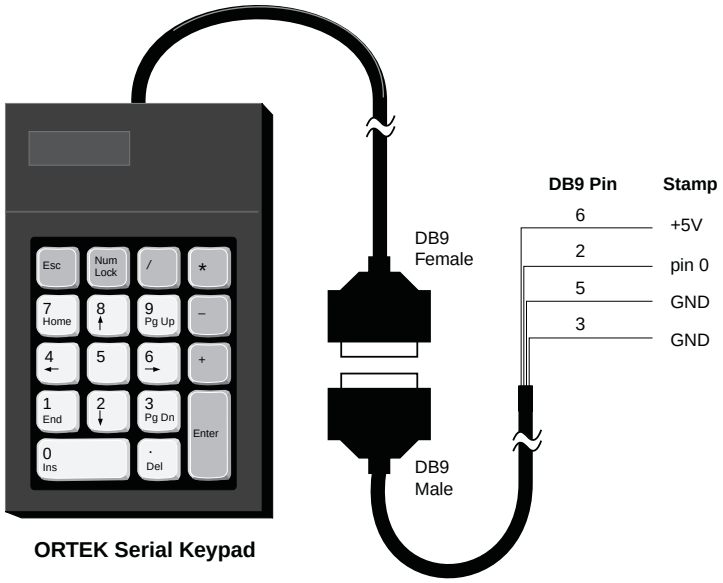

Figure 1. Connecting the serial keypad to the Stamp for data entry requires just one I/O pin plus a few microamps from the Stamp's power supply.

At $49 (see the source list), the keypad isn't cheap, but it's not unreasonably expensive either. So I bought one and proceeded to hack it. Here's what I found:

The heart of the keypad circuitry is a PIC microcontroller, just like the Stamp's own processor. This one runs at a leisurely 38.4 kHz; less than 1/100th the speed of a Stamp. This limits current draw to the 10s of microamps. It also helps the keypad meet FCC limits on radio interference.

When you press one of the pad's keys, its PIC sends a one-byte code (generally a lower-case letter) to the PC via an AT-style DB-9 serial connector. The serial data is sent at 1200 baud, with no parity, 8 data bits, and 1 stop bit (N81). If you hold down a key, the PIC waits a moment, then transmits a string of the same one-byte key code in rapid-fire fashion to simulate the "typeamatic" response of the standard keyboard. When you release a key, the PIC sends a different one-byte code (generally the upper-case version of the key-down code). Table 1 summarizes the codes.

Table 1. Key Codes Used by Unmodified ORTEK Keypad

Key	Key-Down (ASCII)	Key-Up (ASCII)
1	‘ (96)	@ (64)
2	a (97)	A (65)
3	b (98)	B (66)
4	c (99)	C (67)
5	d (100)	D (68)
6	e (101)	E (69)
7	f (102)	F (70)
8	g (103)	G (71)
9	h (104)	H (72)
Esc	i (105)	I (73)
/	j (106)	J (74)
*	k (107)	K (75)
-	l (108)	L (76)
+	m (109)	M (77)
. (point)	n (110)	N (78)
0 (zero)	o (111)	O (79)
Enter	t (116)	T (84)
Num Lock	y (121)	Y (89)

From an electrical standpoint, the keypad is wired to derive power from and send signals to the PC serial port. Its interface generates bipolarity RS-232 signals from the stolen port output voltages. However, the pad will work just fine with a single-ended 5-volt supply, like that of the Stamp. See table 2 for connections.

Table 2. ORTEK Keypad Wiring

DB-9	Color	PC Function	Stamp
2	orange	receive data	serial in
3	yellow	transmit data	ground
6	white	DSR handshake	+5 volts
5	black	ground	ground

Armed with nothing more than the information above, you could make use of the ORTEK keypad for Stamp data entry. Figure 1 shows the hookup, while listing 1 demonstrates how to make sense of the data. However, I couldn't leave well enough alone.

PC Keypad + PIC Program = "Stamp Pad"

The SERIN command can convert strings of ASCII text like "123" into equivalent numbers without the additional programming effort of listing 1. If only the keypad generated terminal-style output...

Instead of wishing, I picked up a soldering iron and removed the keypad's PIC controller. I probed the remaining keypad circuitry and determined the following:

• Like most keypads, this one is wired as a matrix of row and column connections. When a key is pressed, it shorts one row to one column. The processor looks up the row and column coordinates of the short in a table stored in ROM to translate it to a key code.

• The pad's column connections go to the four bits of PIC port RA, which are configured as inputs. (On the Stamp, RA is used to communicate with the PC and manage the EEPROM. It isn't accessible to the user.) When no key is pressed, all bits of RA are pulled high (reading 1s) by four resistors.

• The pad addresses the row connections with

the outputs of a 74HCT138 eight-channel multiplexer. This chip accepts a three-bit address from the lower bits of PIC port RB, and outputs a low on one of its eight outputs. Another bit of RB is used to activate and deactivate the ‘138. Bit RB.7 serves as the serial output. The remaining three bits of RB are unused.

Based on these observations, I coded a replacement PIC in assembly language. Thanks to the clever design of the hardware, this was extremely easy to do. Most of the effort went into preparing tables that unscrambled the row/column coordinate system of the pad, which was apparently designed to permit an inexpensive, one-sided circuit-board layout.

In addition to changing the data format with my new program, I added a hardware feature; a key-acknowledgment beeper. If an inexpensive piezo beeper is connected between pin RB.4 of the PIC and circuit ground, the beeper will make a short “bink” sound to acknowledge each key press. For applications that lack a display to confirm the numbers being entered, this is a tremendous help.

The modified keypad acts like a micro terminal that transmits one ASCII character per keypress (as shown in table 3). The characters correspond to the Stamp’s internal table of the ASCII character set. That is, the following line of Stamp code will respond to the code sent by pressing the “*” key:

```
IF theKey = “*” THEN Asterisk
```

My keypad PIC program has a strict debounce function that requires keys to be pressed and released in a deliberate manner. Speed isn’t compromised, as long as users don’t roll from one key to the other. The first key has to be released before the next will register.

You can make your own Stamp Pad. Figure 2 shows details of the modification. To use the modified pad, connect it to the Stamp as shown in listing 1, then use Serin to collect the data. If, for instance, you want a 16-bit number to be stored to variable w1, use:

```
SERIN 0,N1200,#W1
```

Table 3. Key Codes for “Stamp Pad”

Key	Transmitted Text (ASCII value)
1	1 (49)
2	2 (50)
3	3 (51)
4	4 (52)
5	5 (53)
6	6 (54)
7	7 (55)
8	8 (56)
9	9 (57)
Esc	<ESC> (27)
/	/ (47)
*	* (42)
-	- (45)
+	+ (43)
. (point)	. (46)
0 (zero)	0 (48)
Enter	<RETURN> (13)
Num Lock	<SPACE> (32)

The # symbol before the variable name tells the Stamp to interpret up to five keystrokes as a number ranging from 0 to 65,535 (the range of a 16-bit number). The user must press Enter or any other non-numeric key to finish the entry. If you just want to know which key was pressed, use a byte variable without the #:

```
SERIN 0,N1200,B2
```

After the instruction executes, B2 will contain the ASCII value of the key pressed, as shown in table 3. If you have used SERIN before to accept data from a PC or other computer, this stuff is old hat by now.

You can obtain the Stamp Pad PIC from me, either by buying it outright, or by obtaining the source code free with purchase of my PIC Source Book. This is a collection of assembly language routines for the PIC that mimic the functions of the BASIC Stamp, helping users to move their programs into faster, more efficient PIC hardware. See the source box at the end of this column.

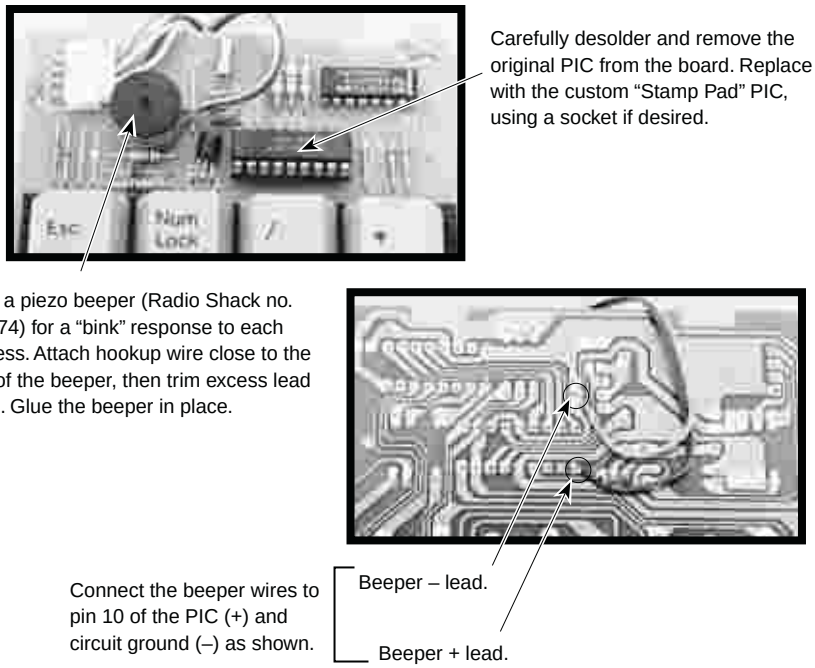

Figure 2. Details of the Stamp Pad modification.

But Wait, There's More!

There's an expression about sausage makers using “every part of the pig except the squeal.” In that spirit, I couldn't ignore the software that came with the keypad. It redirects codes from the serial port to the keypad buffer, allowing you to use numeric input from the pad in any DOS or Windows application.

It occurred to me that if the Stamp were programmed to mimic the key codes, and the keypad software were installed, then the Stamp could “type” data at the keyboard. Think about it--Stamp-collected data magically appears in your spreadsheet, database, or word processor document. No fumbling with terminal software or file transfers.

There are general-purpose software utilities and hardware devices sold for this very purpose, with prices from $100 to $500. They accept any kind of serial input, not just numbers. But if numbers are all you need, check out listing 2. To use the program, install the keypad software according to the instructions that come with the pad. Temporarily disable the keypad driver by typing “KEYPAD OFF” at the DOS prompt. Connect a Stamp programmed with listing 2 as shown in table 4, but don't connect power to the Stamp yet. Next, turn on the keypad driver by typing “KEYPAD” at the DOS prompt.

Table 4. Stamp-to-PC Connections for Listing 2

Pin	DB9	DB25
pin 0	2	3
GND	5	7

You're ready to go. For a test drive, boot the BASIC Stamp host program STAMP.EXE, but don't load a program. Reconnect power to the Stamp. A column of numbers will be typed onto the screen. That's the Stamp, communicating with your keyboard buffer through the keypad software. Now you can write data-acquisition programs for the Stamp that can communicate directly with any piece of PC software you own!

By the way, the reason for the somewhat roundabout setup procedure above is to avoid trashing your work. When I wrote the program in listing 2, I already had the keypad driver resident in my PC's memory, and the Stamp connected to the serial port. As soon as I ran the program, it began typing into my just-finished Stamp program listing. I had to quickly

disconnect the Stamp, and erase all of the numbers it had added to the listing.

Conclusion

If you're interested in other keypad solutions for the Stamp, make sure to get Parallax application note no. 3, which shows how to connect a 74C922 16-key pad and an LCD to the Stamp. Looking for an interesting application for our serial keypad? Try Parallax application note no. 6, a stepper-motor driver that can be controlled directly from the Stamp Pad.

NOTE: This article was originally published in 1995. 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: OR_KEYS.BAS (interpret ORTEK MCK-18S/N keypad codes)
' This program accepts serial data from the ORTEK MCK-18S/N numeric
' keypad and converts it into a 16-bit value in variable w1. Users
' must be careful not to hold keys down, or they will activate the
' pad's autorepeat function, causing entry errors.

' Main program loop.
Loop:
gosub GetKeys              ' Getkeys does all the work.
debug w1                   ' Show the result on the PC screen.
goto Loop                  ' Do it forever.

' Subroutine to receive the serial data, filter out extraneous key-up
' codes, and convert received data bytes to a 16-bit value. GetKeys
' will keep accepting and interpreting digits until the user presses
' a non-numeric key, like <Enter>. The routine takes advantage of the
' fact that the key codes for the numbers 1-9 are sequential;
' subtracting 95 from them leaves you with the number itself.
' Although0 (zero) is out of sequence, an additional IF/THEN
' statement recognizes its code ("o").
GetKeys:
let w1 = 0                 ' Clear w1 to start.
Again:
 Serin 0,N1200,b0          ' Get code from the keypad.
 if bit5 = 0 then Again    ' Bit5 is 0 in key-up codes; ignore em.
 if b0 <> "o" then skip    ' Change 0 code from "o" (111) to 95.
 let b0 = 95
skip:
 if b0 > 104 then done     ' Non-numeric key pressed; we're done.
 let b0 = b0 - 95          ' Otherwise convert to 0-9, multiply
 let w1 = w1 * 10 + b0     ' old total by 10, add new value, and
goto Again                 ' get the next digit.
done:
 return                    ' Done: return to main program.
```

```
' Listing 2: FAKE_PAD.BAS (imitate the ORTEK keypad codes)
' This program mimics the codes produced by the ORTEK keypad,
' allowing a PC running the ORTEK software to receive Stamp data
' to its keyboard buffer. The Stamp "types" directly into programs
' that are incapable of normal serial input. To demonstrate this
' capability, the Stamp will count upward by 1 and type the result
' of each calculation to the PC. Remember that the keypad software
' must be installed on the PC for this to work. Before running
' this program, make sure that this program is saved, since the
' Stamp may begin typing numbers into it, if the keypad software
' is active.

SYMBOL nonZero = bit0      ' Leading-zero suppression flag.
SYMBOL code = b1           ' Key code to send.
SYMBOL decade = w2         ' Power-of-10 divisor for conversion.
SYMBOL count = w3          ' Counter for demo.

' Main program loop.
Loop:
 let w1 = count            ' Transfer value of copy to w1.
 gosub typeData            ' "Type" the data to PC.
 pause 100                 ' Wait briefly
 let count = count + 1     ' Increment counter.
goto Loop                  ' Do it forever.

' Subroutine to convert the value stored in w1 to ORTEK keypad codes.
typeData:
let nonZero = 0            ' Clear flag that indicates first non-0 digit.
let decade = 10000         ' Start with highest digit of w1.
nextDigit:
 let code = w1/decade      ' Get value of current digit.
 let w1 = w1//decade       ' Leave remainder in w1.
 if code=0 AND nonZero=0 AND decade <> 1 then skip3   ' No leading 0s.
 if code=0 then skip1
 let nonZero = 1
 goto skip2
skip1:
 let code = 16             ' Code for 0, minus 95.
skip2:
 let code = code + 95
 serout 0,N1200,(code)     ' Send key-down code of digit.
 let bit13 = 0             ' Clear bit5 of b1 (bit13) for key-up code.
 serout 0,N1200,(code)     ' Send key-up code.
skip3:
 let decade = decade/10    ' Get ready for next lower digit
 if decade > 0 then nextDigit
 serout 0,N1200,("tT")     ' Done. Send <Enter> key.
return
```