Easy GPS Readings for the Basic Stamp Chip

By Randy R. Price
Kansas State University
Biological and Agricultural Engineering Department

Introduction:

Small microcontrollers, such as the Basic Stamp II by Parallax Inc., are popular to students, researchers, and industry designers. These chips provide an easy way for people to build high-tech electronic systems with very little wiring or knowledge about electronics. Still, these chips have not been used to a great extent for data-logging because of problems reading in the GPS data. These problems range from writing the parsing code needed to separate the GPS numbers from text strings; to the coordinates being represented as real numbers (Basic Stamps do not contain variable space for real numbers). For these reasons, developers have been forced to use higher order chips (BS2p, etc.) and certain GPS units that output proprietary statements with less data. This paper introduces a method and code to allow the Basic Stamp II to read in normal NMEA 0183 statements from a GPS at the 1 hertz rate. This code can also be used in other chips for faster GPS reading without parsing.

Literature Review:

GPS units have been around for the last 25 years. During this time, they have become available to the average consumer for an affordable price. The standard output of these units is date, time, location, speed, course, and altitude. The standard format is given by the National Marine Electronics Association (NMEA) and is designated NMEA 0183. The code always starts with a line descriptor (such as $GPRMC) that defines the information that will follow. Sentences are generally outputted every second with anywhere from 1 to 10 sentences for each generation (depending upon the NMEA version and statements selected by the GPS). Versions of the code range from 1.0 to the most current: 3.01 (January of 2002). The $GPRMC statement is one of the most useful statements and gives date, time, location, speed, and course heading in one sentence. Typical GPS locations are outputted in decimal degrees combined with minutes to form a real number with a DDMM.MMM (decimal degrees minutes.minutes) format. Decimal degrees can range anywhere from 0 to 180. An official listing of these codes is found at http://www.nmea.org/pub/0183/index.html.

Most GPS units output the NMEA code at the prescribed rate of 4800 baud, but increases in hardware capabilities have allow higher speeds such as 9600, 19200, and 38400. Before these baud rates were possible, some manufacturers notice that the all the GPS information could not be transmitted in a one second time period (depending upon how many sentences were transmitted), so they included a subset of proprietary commands that allowed the user to output only a certain number of sentences, or even proprietary statements with certain pieces of information. These statements could be

outputted with or without the standard NMEA code statements. Some GPS units output the numbers in fixed width format and some do not[1].

Several authors have noted the problems reading GPS units with Basic Stamps. These problems range from non-fixed width sentences to the amount of data transferred and baud rate of the GPS. Jon Williams (2003) states the following about creating a GPS data-logging program for an airplane. "Ken wanted to track his (RC) plane's flight path and speed, and asked me to come up with a method of doing it. The solution was to strap a small GPS receiver (Garmin® eTrex) onto the plane with a BS2p acting as a data logger. I wrote about the methods used in our airplane data logger back in March of 2002. The program has served us well, but Ken has been asking for better resolution in the data. You see, the old program uses standard NMEA 0183 strings from the GPS receiver that are spit out at 4800 baud. With the bulk of information dumped by the receiver at this baud rate we only get updates every two seconds. For a model airplane traveling around 80 mph, this isn't great. What to do? While reviewing the eTrex manual I found that it has a simple text (proprietary Garmin®) output method that can be set to 9600 baud. This is a good start as it doubles the communication speed. The other nice thing about this method is that it uses fixed-position fields. This will help make parsing data easier as we know exactly where everything is within the string. When using the $GPMRC string, some fields are variable-width which complicates the location of data". Still parsing itself can take quite a bit of processor time since the code itself must be first read in from GPS (using the GPRMC statement), stored in a scratch pad ram, and then broken down into individual numbers. To reduce the breadth of this paper, I am not going to list the parsing code from above, but it can be found on the parallax website (www.parrallax.com). I do give the memory map for the code (in a BS2p chip) in Figure 1. This memory map shows the code using approximately 76% of the BS2p's memory and more than 10 of the 12 word size variables. This code does not leaving much variable space for other data-logging operations. Note, though, that this code is doing quite a few number evaluations (such as giving the true date, time, etc.) and could be reduced for smaller, more specialized applications. In either case, the BS2 chip could not run the code because it does not have the scratch pad ram (SPSTR) needed for storing the NEMA 0183 sentence.

Objective:

The objective of this paper is to develop a code that will allow the BS2 chip and other small microcontrollers to read in the NMEA 0183 data from a 4800 baud GPS at the 1 Hz rate for the purpose of data-logging.

Procedure:

Basic Stamps are very good at reading serial inputs and come with many utilities. Several of these utilities are the WAIT and SKIP commands, and the DEC variable modifier. The wait command allows the BS2 chip to wait on a certain NMEA 0183 output sentences (such as the "GPRMC' statement) and the skip command allows the chip to skip non-essential information such as a comma and letter. The DEC command

[1] Fixed width sentences transmit the leading zeros which causes the numbers and data to always reside in the same column within a sentence. This quality allows easier parsing of the sentence since the numbers always lie in known locations.

tells the chip to read in a decimal number (not a text string) and as will be discussed later, allows the chip to output a fixed format number.

For a data-logging application, the GPS coordinate must be read into the device and then combined with some other piece of information such as voltage or temperature. This data is then usually outputted to some other device (such as a desktop computer or memory card), where it is plotted or stored for later analysis. Because the numbers are only collected and sent elsewhere (and not used in calculations) the problem and code becomes much easier for the BS2.

In our problem, the GPS coordinates, although real, can be read in as two separate decimal numbers and then recombined on output to resemble the original number. This operation is possible because the Basic Stamp thinks the decimal point between the real number is the same as a comma (the Basic Stamp treats all delimiters - space, comma, period, etc. – exactly the same – a delimiter between numbers). Also, the Basic Stamp word size variable can only hold a number up to 65535. The largest real part of an east-west coordinate is 18000. Typically this number is outputted with four decimal places of precision, so the largest fractional part of the coordinate is 9999. The Basic Stamp can hold each of these numbers in a word size variable. Once the coordinates are read in as integers, the numbers can then be outputted (serially) with a "." between the two numbers to create an original real number. Using this method, the receiving device thinks it received a real number, even though Basic Stamp can't theoretically hold a real number.

One problem still exists though. The leading zeros in the fractional part of the decimal number will be truncated by the chip. For instance, the number 102.00345 becomes the two integers 120 and 345. When put back together on output (using "102" + "." + "345"), the output becomes 120.345, which is not the original number. Luckily, the Basic Stamp has a way to solve this problem. GPS units always output at a certain number of decimal precision (usually 4 units) that stay the same. Because of this fact, the fractional number should always have this same precision. Also, the DEC formatter will create leading zeros and when combined with the original digits of precision, will allow output of the correct fractional number. For instance, if the original precision was 3 digits, and the fractional integer is 10, then setting the DEC formatter to DEC3 will output 010 (keeping the same digits of precision). If the integer for the fractional part is 789, and the digit of precision is 3, then the output would be "789". Using this method, the correct decimal part of the fractional number can be outputted correctly, allowing the Basic Stamp to read in the GPS coordinates and output them correctly.

A code was written for a Basic Stamp following the principles outlined above. The first code (Table 1) was written using the DEC4 modifier in the output statement. The second code (Table 2) was written if the DEC4 modifier wasn't available, and IF THEN statements were used to output the correct number (this code is good for different processors without the DEC4 function). The code reads in time, location coordinates, speed, and course from the "GPRMC" statement and then outputs the data to the screen[2]. The "serial 16,n9600,[ ]" was substituted for the typical DEBUG statement so that the output can be received by any RS-232 receiving program, such as HyperTerminal® in

[2] Note that the time value is not the exact time given by the GPS since a word size variable cannot hold this number. This number can still be used though to evaluate the elapsed seconds. Also note that the location directions (E/W and N/S) were inserted automatically in the output statement and not read from the GPS chip. These values do not change for most locations, but must be set correctly for your particular location.

Windows® (usually found in the Accessories/Communications folder of the Start Menu). The HyperTerminal® program can be useful for capturing data into a text file for later plotting in the Excel® spreadsheet or some other program[3].

The code was testing on a BS2 chip (rev. C) using a Garmin® 15-H OEM GPS. This GPS engine is similar to the engines used in standard handheld GPS units. The GPS RS-232 output was read into pin 0 of the BS2 chip by using a 15k ohm resistor in series with input line to reduce the standard +/- 12 volts of the RS-232 line driver down to +5 volts (to prevent damage to the BS2 chip).

For most data-logging purposes, the program must return to the GPS statement before the next GPS output occurs and still read some other parameters, such as voltage from an A/D (analog to digital) chip or a temperature sensor. This property will allow the chip to be used for data-logging purposes other than just location recording. For this reason, readings were taken to see how much time was left by the code to run other statements. This time was determined by using a PAUSE statement between the input and output statements, and adding time to the pause statement, until the chip could no longer read the next consecutive GPS statement.

Results

The results for the tests are shown in Table 3 and 4. These outputs show time, location, speed, and course of the GPS as captured by the BS2 chip in different locations. In all trials, the time showed consecutively recorded seconds, indicating that the BS2 chip read the GPS output at the 1 Hertz frequency. In table 3, the first eight rows were for a stationary GPS unit and the next eight rows were for a steady walk in a south direction. In Table 4, the results show the GPS readings approximately ¼ mile south-west of the data from Table 3, where the fractional part of the latitude is less than 0.1, showing that the leading zeros were inserted correctly in the output.

The amount of memory and variable space used by the code is shown in Figures 2 and 3. These results indicate that the code used either 7% (DEC4 version) or 16% (non DEC4 version) of the EEPROM space and 8 of the 12 word size variables. Also, testing with the code determined that the time between readings was 0.780 seconds for the DEC4 code and 0.750 second for the non-DEC4 code. These times leave sufficient duration to read other devices such as voltages from an A/D chips or temperature readings from a sensor.

Conclusion:

A code was developed to read in GPS data from a NMEA 0183 statement using the BS2 chip. This code will also work with other micro-processors. The code allowed the chip to read in the coordinates, time, speed and course heading at the 1 hertz GPS output rate. The code only used 7% of the BS2 chips memory and approximately 0.75 seconds were left to perform other operations, such as reading a voltage A/D chip or a temperature sensor. The code also allows different GPS receivers to be used without changing the code since the code is NMEA 0183 compliant.

[3] Note that the typical debug window in the Basic Stamp Software Package will still work with this statement, but may have to be started manually. I have inserted a DEBUG statement in the noGPS sub-routine part of the program that will automatically start the Debug window for you.

Table 1
Program to Read in the “GPRMC” statement from a GPS receiver with the DEC4 modifier
gps reader.bs2

```
' '{$STAMP BS2}
'By Randy Price - March 2002
'Program to read in GPS coordinates from the NMEA 0183 GPRMC statement

gpstime  VAR   Word
gpstime2  VAR  Byte
N  VAR   Word
W  VAR   Word
NN   VAR   Word
WW   VAR   Word
speed1   VAR   Word
speed2  VAR  Nib
course1  VAR  Word
course2  VAR  Nib

'Baud rates:
'16572 = 4800 baud BSII Chip
'16468 = 9600 baud BSII Chip
n9600 CON 16468
n4800 CON 16572

main:

'Get next GPS RMC statement at 4800 baud pin 0
SERIN 0,n4800,2100,noGPS1,[WAIT("RMC,"),DEC gpstime, SKIP 3, DEC N, DEC NN, SKIP 3, DEC W, DEC WW, SKIP
3, DEC speed1, DEC speed2, DEC course1,DEC course2]

PAUSE 780

GOSUB output1
GOTO main

'****************************************** OUTPUT ROUTINE ********************************************
output1:

'output to terminal at 9600 baud
SEROUT 16,n9600,[DEC gpstime,",",DEC W,".",DEC4 WW,",W,",DEC N,".",DEC4 NN,",N,",DEC speed1,".",DEC
speed2,",",DEC course1,".",DEC course2,10,13]
RETURN

nogps1:
DEBUG "no gps",CR
GOTO MAIN
```

Table 2
Program Code to Read in the “GPRMC” statement from a GPS receiver without the DEC4 modifier
gps reader 2.bs2

```
‘{$STAMP BS2}
'By Randy Price - March 2002
'Program to read in GPS coordinates from the NMEA 0183 GPRMC statement

gpstime  VAR   Word
N  VAR   Word
W  VAR   Word
NN   VAR   Word
WW   VAR   Word
speed1   VAR   Word
speed2  VAR  Nib
course1  VAR  Word
course2  VAR  Nib

'Baud rates:
'16572 = 4800 baud BSII Chip
'16468 = 9600 baud BSII Chip
n9600 CON 16468
n4800 CON 16572

main:

'Get next GPS RMC statement at 4800 baud pin 0
SERIN 0,n4800,2100,noGPS1,[WAIT("RMC,"),DEC gpstime, SKIP 3, DEC N, DEC NN, SKIP 3, DEC W, DEC WW, SKIP 3,
DEC speed1, DEC speed2, DEC course1,DEC course2]

GOSUB output1
GOTO main

'****************************************** OUTPUT ROUTINE ******************************************
output1:

'output to terminal at 9600 baud
SEROUT 16,n9600,[DEC gpstime,",",DEC W]

IF WW < 10 THEN gps1
IF WW < 100 THEN gps2
IF WW < 1000 THEN gps3
SEROUT 16,n9600,[".",DEC WW]
GOTO gpsNN

gps1:
 SEROUT 16,n9600,[".000",DEC WW]
GOTO gpsNN

gps2:
 SEROUT 16,n9600,[".00",DEC WW]
GOTO gpsNN

gps3:
 SEROUT 16,n9600,[".0",DEC WW]
GOTO gpsNN

gpsNN:
 SEROUT 16,n9600,[",W,",DEC N]

IF NN < 10 THEN gps4
IF NN < 100 THEN gps5
IF NN < 1000 THEN gps6
SEROUT 16,n9600,[".",DEC NN]
GOTO gpsend

gps4:
```

```
 SEROUT 16,n9600,[".000",DEC NN]
GOTO gpsend

gps5:
 SEROUT 16,n9600,[".00",DEC NN]
GOTO gpsend

gps6:
 SEROUT 16,n9600,[".0",DEC NN]
GOTO gpsend

gpsend:
 SEROUT 16,n9600,[",N,"]

'End Data output from GPS
SEROUT 16,n9600,[DEC speed1,".",DEC speed2,",",DEC course1,".",DEC course2,10,13]

RETURN

'************************************* GPS Not Detected Routine ***************************************
'(having the Debug statement here will cause the debug window to come automatically but not interfere with data collection in
HyperTerminal)

nogps1:
DEBUG "no gps",CR
GOTO MAIN
```

Table 3
Typical Text Output from Source Code [ab]

Time	Longitude	Direction Indicator	Latitude	Direction Indicator	Speed (knots)	Course (degrees)
34648	9634.9778	W	3911.3329	N	0	0
34649	9634.9779	W	3911.3329	N	0	0
34650	9634.9779	W	3911.3329	N	0	0
34651	9634.9778	W	3911.3329	N	0	0
34652	9634.9778	W	3911.3329	N	0	0
34653	9634.9779	W	3911.3328	N	0	0
34654	9634.9779	W	3911.3328	N	0	0
34655	9634.9779	W	3911.3329	N	0	0
34656	9634.9782	W	3911.3328	N	0.8	0
34657	9634.9795	W	3911.3311	N	1.6	195.1
34658	9634.9798	W	3911.3306	N	1.6	195.1
34659	9634.9794	W	3911.3301	N	1.9	177.3
34660	9634.9791	W	3911.3296	N	1.9	177.3
34661	9634.9789	W	3911.3291	N	2	176
34662	9634.9788	W	3911.3286	N	2	176
34663	9634.9789	W	3911.328	N	2	176

[a] Data was imported into the Excel® spreadsheet and formatted to fit the individual columns.
[b] The column headings were not part of the original file and were added in for descriptive purposes.

Table 4
Typical Text Output from Source Code [ab]

Time	Longitude	Direction Indicator	Latitude	Direction Indicator	Speed (knots)	Course (degrees)
53329	9635.0191	W	3911.2532	N	1.8	172.6
53330	9635.0192	W	3911.2526	N	2	178.6
53331	9635.0191	W	3911.252	N	1.9	176.4
53332	9635.0191	W	3911.2514	N	2	179.8
53334	9635.0189	W	3911.2503	N	2	173
53335	9635.0188	W	3911.2497	N	2.1	174.5
53336	9635.0188	W	3911.2492	N	2	177.1
53337	9635.0189	W	3911.2487	N	2	178.8
53338	9635.0188	W	3911.2482	N	1.9	177.4
53339	9635.0188	W	3911.2477	N	1.9	178.6
53329	9635.0191	W	3911.2532	N	1.8	172.6
53330	9635.0192	W	3911.2526	N	2	178.6
53331	9635.0191	W	3911.252	N	1.9	176.4
53332	9635.0191	W	3911.2514	N	2	179.8
53334	9635.0189	W	3911.2503	N	2	173
53335	9635.0188	W	3911.2497	N	2.1	174.5

[a] Data was imported into the Excel® spreadsheet and formatted to fit the individual columns.
[b] The column headings were not part of the original file and were added in for descriptive purposes.

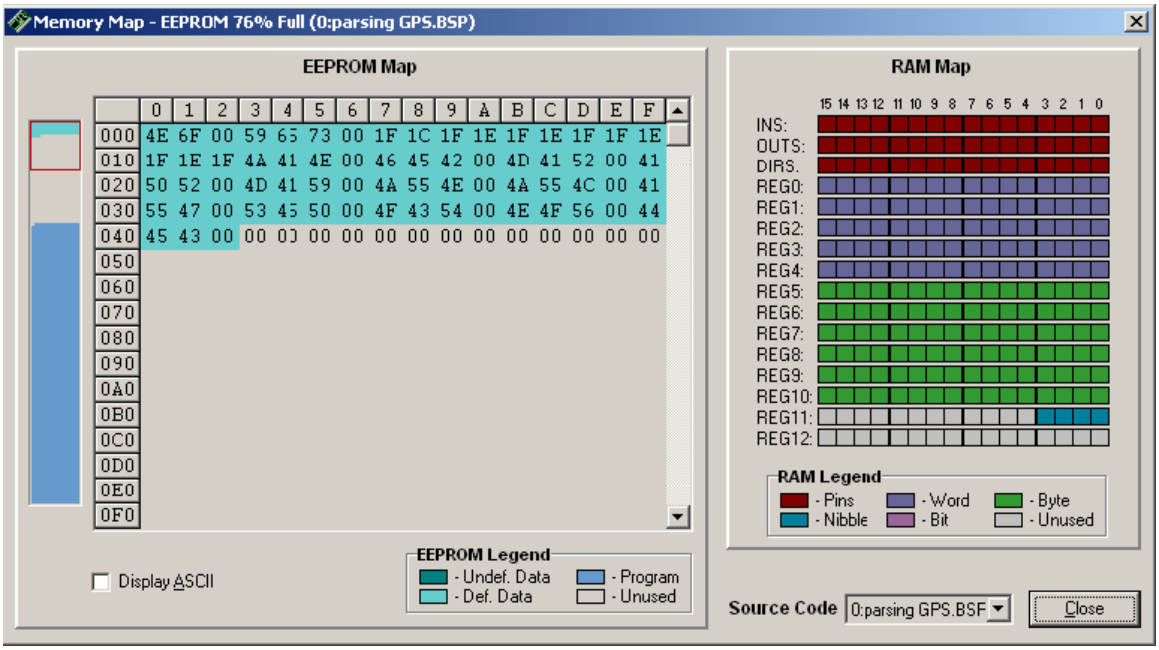

Figure 1: Memory map for BS2p chip using the parsing code.

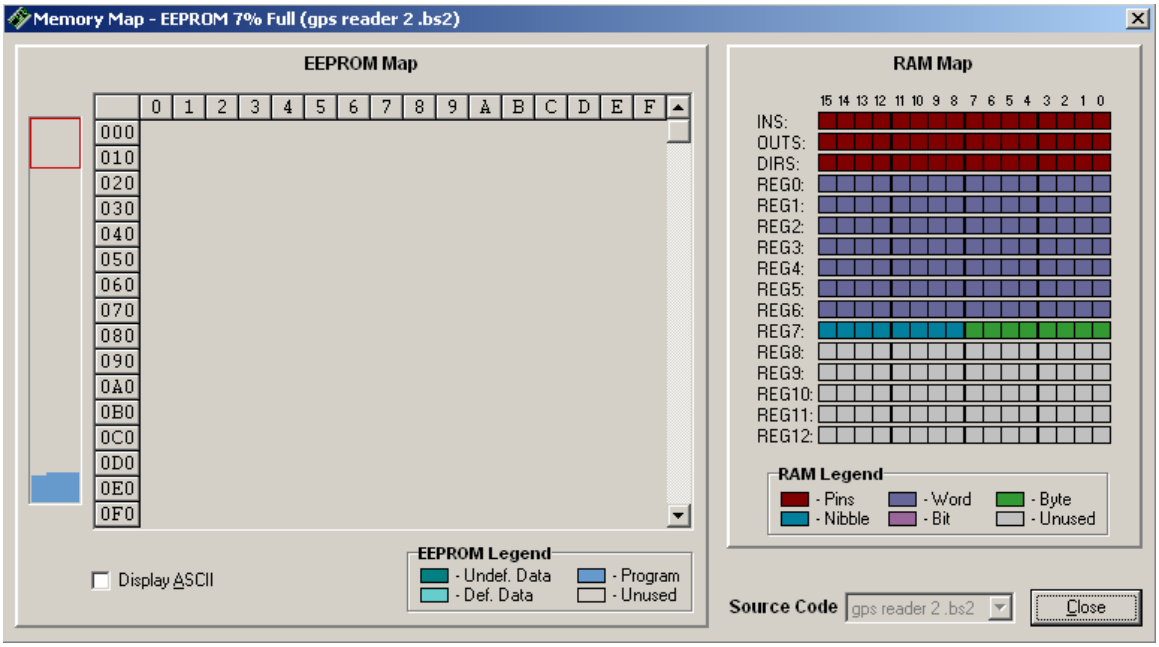

Figure 2: Memory map for a BS2 chip using the non-parsing GPS program and the DEC4 modifier.

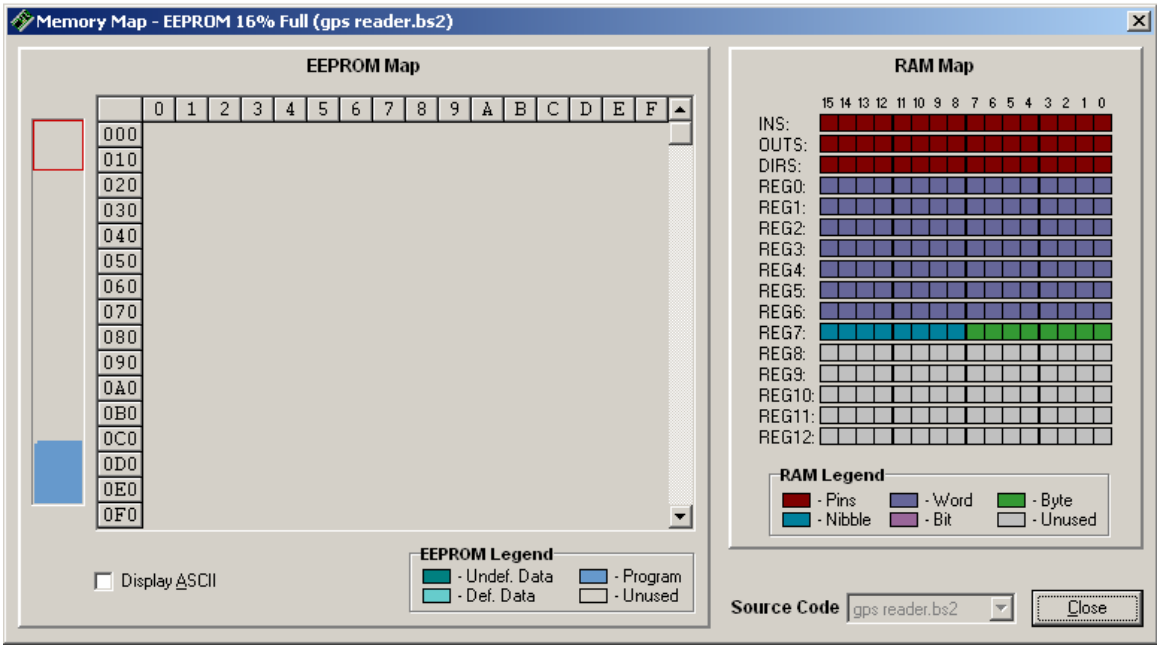

Figure 3: Memory map for a BS2 chip using the non-parsing GPS program without the DEC4 modifier.

References:

Jon Williams, 2003, *"Stamping on Down the Road"*, The Nuts and Volts of BASIC Stamps, Column #103, Volume 4, pp 175-194.

Jon Williams, 2002, *"Where in the World is My Basic Stamp"*, The Nuts and Volts of BASIC Stamps, Column #83, Volume 3, pp 119-137.