TXTFOR.BA
Text Formatting Program for NEC PC-8201A and TRS-80 Model 100

By

Lloyd Johnson

1.0 Introduction

Years ago, I worked in an office that had two PCs that were shared among 15 or so engineers. I brought my personal NEC PC-8201A to work to write status reports and using the built in TEXT program. I manually formatted the report by putting carriage returns where needed. The report was printed and revised a few times before turning it in. If text had been inserted or deleted, it was necessary to rearrange the carriage returns so the right margin would line up correctly. This was a pain.

To solve this problem, I wrote a simple text formatting program that would split and recombine lines as necessary so as to approach but not exceed the maximum number of characters I had specified for a line. I used dot commands to change the program parameters as necessary. I also added new features from time to time.

Unfortunately, this program did not survive the many years of inactivity my NEC PC-8201A has experienced. In fact, it did not look like the NEC itself was going to survive. However, after replacing the NiCad batteries and allowing the internal battery charge, the NEC is now working.

I decided to recreate the text formatting program I had written many years earlier. To demonstrate its functionality, the documentation you are reading was processed by this text formatting program. The name of this program is TXTFOR.BA

2.0 Operation

After executing the program, a list of the files is displayed and the user is requested to enter an input file. A .DO file name should be entered without the .DO extension. The user is then requested to enter the output name. If nothing is entered, an output of "COM:" is assumed. It is assumed the RS-232 parameters have already been established by the TELCOM program. If it is desired to set the parameters with this program, then an output of "COM:5I71XS" can be entered for the NEC or "COM:57I1E" for the TRS-80 Model 100. With the TRS-80 Model 100, an entry of just "COM:" is not accepted and you are forced to enter "COM:57I1E".

The TXTFOR.BA program processes dot commands embedded in the .DO input file. A dot command is a command beginning with a period or some other specified character and is followed by one or two letters and possibly a numeric parameter. It is important to leave a space between the letters and the numeric parameters for the command. These commands control the text formatting process. If the .DO file does not contain any dot commands, the .DO file is transferred to the output without formatting. I quite often use this program to transfer BASIC programs to my desktop machine. The BASIC programs have first been saved as a .DO file. As a BASIC program, all lines begin with a number so there are no dot commands and the transfer is done without formatting.

The available dot commands are described in section 4 of this document.

The input .DO file should have a carriage return every four to six lines. Six lines of text at 40 characters per line is 240 bytes. This is getting dangerously close to the maximum string length for BASIC. If you encounter an OS error, you may need to add more carriage returns to your input .DO file.

The TXTFOR.BA program does not generate any error messages. Although some illegal inputs are ignored, others may result in BASIC error messages or other undesirable results. Perhaps a later version of this program will be more robust and bullet proof, but don't count on it.

3.0 Program Organization/Description

The first 15 percent or so of this program consists of three sections which are: Initialization, Main and End Program. The remaining 85% of the program is subroutines. The initialization section sets the default values for some parameters, dimensions an array and retrieves input and output file/port names from the user. The main section reads lines of text from the input file and calls a subroutine to process it. When an EOF (End of File) is encountered, the program passes control to the End Program section where any unprocessed text is sent to the output file/port. Both the input and output are then closed and the program terminates nominally.

There are fifteen subroutines used in this program and all but two are called from other subroutines. After reading a line of text, the Main section calls the Input Processing subroutine. The End Program section has a call to the Output Line subroutine to send any unprocessed text to the output. The Output Line subroutine is also called by other subroutines as necessary.

The Input Processing subroutine checks the input string for dot commands and calls the Process Dot Commands subroutine if one is found. If the center flag (CF) is set, the Center Text subroutine is called. Otherwise, the Fill Flag (FF) is checked. If the Fill Flag is clear, then text is sent to the Output Line subroutine and the Input Processing subroutine returns to the Main section where it was called.

Fill processing takes place in the Input Processing subroutine when the Fill Flag (FF) is set. Fill processing consists of appending the input line to the output line adding a space as necessary. The Tab Stop variable (TS) is checked and the Process Tab Stop subroutine is called when TS is greater than zero. The length of the output line is checked. If it is less than the maximum width (MW) than the subroutine returns to the Main section so more text can be read. If the length is greater than the maximum width, the output line is split at the first space found on the right. The left portion of the output line is printed by a call to the Output Line Subroutine. The right most portion is assigned to the input string variable and control is passed to the beginning of the Input Processing subroutine to continue processing the input.

The Output Line subroutine first trims leading and trailing spaces. A call is then made to the Calculate Indent Amount subroutine to determine how many leading spaces should be added due to the left margin and paragraph indent. The paragraph indent only comes into play if the new paragraph (NP) flag is set. The Justify

Flag (JF) is checked and if set, the Justify Text subroutine is called. The output line is printed and the New Paragraph flag (NP) is cleared. The subroutine then returns to where it was called.

The Justify Text subroutine will right justify the text by randomly adding spaces between words such that the right margin is perfectly aligned. When this subroutine is called, the spaces in the line are counted. The location of each space within the line is stored in the array, SA. One of these locations is chosen randomly and a space is added to the line at that location. The locations in the array after that location are updated and this process is repeated until the length of the line is equal to the maximum width variable, MW.

The operation of the other subroutines is fairly straight forward and will not be described in this document.

4.0 Dot Commands

In this section, the dot commands are listed and briefly described. As previously mentioned, a dot command is a line beginning with a period or another specified character. I initially planned to use just a period, but in order to document the dot commands in this document, a different character was required. Changing the character is done by the CC command which is discussed later. Here is the list of dot commands in no particular order:

.F	Fill command. This turns on the word fill process by setting the Fill Flag variable (FF) to 1. An input line will have text from the next line appended to it. The resulting line will be split at a space such that the line length is as close to the maximum allowed line length as possible without exceeding it.
.NF	No Fill. This command clears the Fill Flag (FF) by setting it to 0. With FF=0, text is sent to the output without any further formatting.
.P	New Paragraph. When this dot command is processed, the New Paragraph flag (NP) is set to 1. Any text remaining from previous fill processing is sent to the output and extra lines are printed based on the value of PS (Paragraph

Spacing).

.P n	Paragraph spacing where "n" is the number of extra lines to print. This command changes the value of the Paragraph Spacing variable (PS). It also invokes the new paragraph dot command.
.PI n	Paragraph Indent. This command sets the value of the Paragraph Indent variable (PI) to n. The new paragraph dot command, P is also invoked by this command.
.SM n	Set Maximum characters. This command sets the Maximum Character variable (MC) to the value n. The variable, MC represents the maximum characters allowed on a line.
.LM n	Left Margin. This commands sets the Left Margin variable (LM) to n. The variable LM represents the number of spaces that will be added to the beginning of the line prior to printing it.
.J	Justify. This command will turn on text justification by setting the Justify Flag (JF) to 1. The Fill Flag (FF) is also set to 1. Note: This command will significantly increase the text processing time.
.NJ	No Justify. This command clears the Justify Flag (JF) by setting it to 0.
.C	Center Text. This command sets the Center Flag (CF) to 1. The Justify Flag (JF) is also set to 0 by this command.
.NC	No Center. This command clears the Center Flag (CF) by setting it to 0.
.TS n	Tab Stop. This sets the variable for the Tab Stop (TS) to the value of n. This program currently only allows for one tab stop. Tab stops are only processed if the Justify Flag (JF) is 0.
.CC	Change Command character. There are certain cases where it is necessary to change the command character to something other than a period (this document for example). This command was developed

for that purpose. With this command the command character can be changed from a period to some other character such as an asterisk (*). commands. Since I'm more used to entering dot commands than star commands, I typically will use this command to change the command character back to a period when it is no longer necessary to be an asterisk.

5.0 Variable Description

I will attempt to give a brief description of the variables used by this program. This list was put together manually so I apologize in advance for any errors or omissions.

A variable list is invaluable if you are to modify the program to add features; modify it to take less memory; or heaven forbid, modify it to correct a bug.

So without further ado, here are the variables:

CC$	Command Character. This variable contains the character that identifies the line of text as a command when this character appears at the beginning of a line. The default value of this character is a period, ".". This value can be changed by the dot command, CC.
CF	Center Flag. When this variable is set, spaces are added to left of the sent line variable, SL$ such that the text is centered within the line. This flag is set by the dot command, C and cleared by the dot command, NC.
CS$	Command String. This string represents a dot command after the left most character has been removed.
D$	Document name to be formatted. This is the name of the .DO file that will serve as this program's input.
ES	Extra Space. This variable represents the number of extra spaces that are required as a result of a tab stop.

FF	Fill Flag. The default value of this variable is 0. When this variable is set to 1 by a dot command, the Input Processing subroutine will perform fill processing by filling the output line with additional text from the input until the length of the output is as close as possible, but doesn't the maximum width variable, MW.
I	Index. This variable is used in FOR loops as the loop index.
IA	Indent Amount. This is the number of spaces to be output prior to outputting the text, SL$. It is calculated from the value of variables, LM and PI.
JF	Justify Flag. When this flag is set, extra spaces will be inserted randomly at the existing space locations in a line such that the right margin is justified (aligned). This flag is cleared and set by dot commands.
LM	Left Margin. This represents the number of spaces that will be inserted before the output text. The actual number of spaces might be modified further by paragraph indenting. The default value of this variable is 0.
MC	Maximum characters allowed per line. This variable can be adjusted via the dot command, SM. The default value for MC is 50.
MW	Maximum Width. This is the maximum number of characters allowed on the line after it has been adjusted by the paragraph indent variable, PI.
NP	New Paragraph flag. This flag is set by the paragraph dot commands. This lets other subroutines know that the spaces on the left may be increased or decreased by the paragraph indent variable, PI. The NP flag is cleared by the Output Line subroutine.
OF$	Output File. This is the name of the output file or port where the program will send the formatted text. If the user presses Enter without entering a name, a default value of "COM:" will be

	assigned to this variable thus redirecting the output to the serial port.
OL$	Output Line. This string variable serves as the string buffer. It initially starts out empty but later has the input string appended to it. Text that is sent to the output is removed from the left side of this variable.
PI	Paragraph Indent. The value of this variable is set by the dot command, PI. This value along with the value of LM are used to compute the indent amount (IA) prior to sending the text to the output.
PS	Paragraph Spacing. This variable controls the number of extra lines to be output prior to starting a new paragraph. The default value of this variable is 0. It can be adjusted by the dot command, P n.
S1$	String 1. This string represents the left side of SL$ when spaces are added during the Justify Text subroutine.
S2$	String 2. This string represents the right side of SL$ when spaces are added during the Justify Text subroutine.
SC	Space Count. This variable represents the total number of spaces that were located in the text line.
SI	Space Index. This is the index into the SA array which identifies the location in the output line where spaces are located.
SL$	Sent Line. This is the line sent to the Output Line subroutine. It is a different variable than OL$ so that margin and indent operations can be performed on SL$ prior to printing.
SS	Split Space. This variable represents the location in the output line where an extra space will be added in the Justify Text subroutine or where the line will be split in the Input Processing subroutine.

SA	Space Array. This array represents the locations of the first ten spaces that were located in a text line. This array gets updated when a space is added to the output line.
T$	Text input. This variable contains the line of text as read from the input file.
TJ	Temporary Justify. This variable stores the value of the Justify Flag, JF so JF can be temporarily set to 0 when the last line of a paragraph is printed. After printing, JF is returned to the original value that was stored in TJ.
TP	Tab Position. This value of this variable represents the position within the string variable, OL$ where the first tab character was found.
TS	Tab Stop. This variable contains the value of the tab stop set by the dot command, TS.

6.0 Conclusion

I hope this program and the documentation are of use to you. It is submitted to the public domain and you have my permission to modify it and distribute it as you wish. I do request that my name be retained in the documentation and the program comments. Enjoy.