BASIC Precompiler User's Manual

Index

1. INTRODUCTION

Technical Systems Consultants' BASIC PRECOMPILER (PC) and EXTENDED BASIC PRECOMPILER (XPC) are designed to give the BASIC user greater flexibility in writing programs, easier to read programs, and smaller 'compiled' files. The precompilers accept BASIC source files and convert to BAsic Compile files (.BAC) similar to the COMPILE command in BASIC. This should not be confused with BASIC compilers that generate machine language code, because PC and XPC generate code that can only be used with BASIC. This is the manual of the precompiler for both BASIC and EXTENDED BASIC and for both the 6800 and 6809.

PC and XPC execute in two passes and will accept any size file on the disk for input as long as enough memory is available for the symbol table. Two types of output can be generated. The first one is a source listing of the basic program complete with line numbered statements and any error messages. The second is the compiled version of the program ready to be executed by BASIC.

It is assumed that the reader is familiar with BASIC so detailed programming examples are not given nor is the syntax of the BASIC language explained.

1.1 Conventions

As in the BASIC User's manuals the following conventions will be used. The statement or command being described will be printed in capital letters. Angle brackets (<>) will be used to enclose essential components of the statement. Square brackets ([]) will be used to enclose optional components.

<essiental item>
[optional item]

1.2 Definitions

LOGICAL LINE
A logical line in BASIC can consist of one or more physical lines. It may also consist of one or more basic statements separated by either a back slash ('\') or a colon (':').

PHYSICAL LINE
A physical line is defined to be one line on a terminal ending with a carriage return. It also can be thought of as one line in the editor.

LETTERS
The set of letters consists of the characters 'A' through 'Z' and 'a' through 'z'. The underscore character ('_') is valid as a letter also.

NUMBERS
The set of numbers consists of the characters '0' through '9'.

HEXADECIMAL NUMBERS
Hexadecimal numbers will start with a dollar sign ('$') and be followed by four characters from the set of numbers and the set of letters 'A' to 'F'.

SEPARATORS
The set of separators consists of any character that is not a letter, number or an underscore.

2. GETTING THE SYSTEM STARTED

Since there are not any built-in editing functions in the compilers -- you must have a previous created source file on disk before using PC or XPC. The source must must be a standard FLEX source file which is simply textual lines terminated with a carriage return. There should be no control characters in the source file except for the carriage return.

THE FLEX COMMAND LINE

The syntax for calling PC or XPC is as follows:

PC <source file> [<compile file>] [+<options>]

or

XPC <source file> [<compile file>] [+<options>]

Where all file names are standard FLEX file names and default to the current working drive. The <source file> is a previously edited file containing the BASIC source lines, <compile file> is optionally specified as the name of a file to contain the result of the compilation, and <options> is a list of options to be invoked when the compiler is called.

The source file by default has an extension of .TXT as if it were generated by the Text Editor. Optionally one can specify a different extension (ie .BAS) or a drive other than the working drive (ie. 1.PROG) by just including it in the file name. For example:

```
+++PC PROG        will default to PROG.TXT on the working drive

+++PC PROG.BAS.1  over rides all filename defaults
```

The binary file, if not specified, defaults to the same name as the source file but with the .BAC extension. Optionally, one may specify a different name by just including it in the command line. For example:

```
+++PC PROG1.BAS P
```

will use as source PROG1.BAS on the working drive and write the binary to P.BAC also on the working drive. If the file P.BAC already exists on the working drive PC will ask the user if the file should be deleted before starting. For example:

```
+++PC PROG1.BAS P

DELETE OLD BINARY (Y/N)?
```

If N is typed after the prompt, PC will quit and return to FLEX. If Y

is typed, the file will be deleted and PC will continue on its way. If anything else is typed you will be prompted again for either Y or N.

Optionally, one may include compiler options on the command line. The list of options must start with a plus sign ('+'), may not have any imbedded spaces, and contain the following letters in any order.

B Do not create the binary file. No binary file will be created even if a binary file name is specified. This is useful when compiling a program to check for any syntax errors or for a listing of the program.

L Suppress the source listing. If not specified, the compiler will print each line as it is read in pass two. Any lines containing errors will automatically be printed regardless of whether or not the option is specified. Also, imbedding a compiler $LIS or $NOL option into the source will turn on or off the source listing.

S S has no meaning whatsoever. It is included just to be compatible with the ASMB command.

N Turn off line numbers. By default, line numbers are printed at the beginning of each logical line. But if the source program already has line numbers, like a normal basic program, more line numbers would only be confusing.

Y Yes, go ahead and delete the binary file. If binary is being generated and the file already exists then specifying Y will over ride the prompt to delete the file.

Some examples:

```
+++PC PROG1 +LY       no listing, delete old binary

+++PC PROG1.BAS +NB   listing on, no line numbers or binary

+++PC PROG1 T +SLY    no listing, binary on T.BAC on the work drive,
                      automatically delete existing binary file
```

The compilers do not have a built-in method to output to a hardcopy device. However, this operation is available through I/O redirection in FLEX. For example:

```
+++P PC PROG1 +B
```

would cause the listing to be sent to whatever device was defined by the P command in FLEX. For further details of the P command see the FLEX User's Manual and the FLEX Advanced Programmer's Guide.

3. FEATURES

Four things stand out as the main features of PC and XPC. They are:

(1) unlimited length variable names;
(2) unlimited length label names;
(3) continuing logical lines across physical line boundaries;
(4) pagination control for listings.

Each feature will be discussed in detail below.

3.1 Variable Names

Variable names may be of any length and may contain letters, numbers, and the underscore character ('_'). The name cannot be a keyword (see appendix A for list of keywords), the first character must be a letter or an underscore, and the name must be terminated by a blank or separator. Also the name cannot start with the letters 'FN' unless it is a call to or definition of a USER DEFINED function. Lower case letters are automatically mapped into upper case, therefore the name 'lower' is the same as the name 'LOWER'. Some examples of variable names are:

```
THIS_IS_A_VARIABLE_NAME
SO_IS_THIS
THIS_IS_A_STRING_VARIABLE$
_SO_IS_THIS_$
this_is_too$
FUNCTION_IS_A_FUNCTION_NAME
_THIS_IS_AN_INTEGER_VARIABLE%
_SO_IS_THIS_%
THIS_IS_A_DUPLICATE_NAME
this_is_a_duplicate_name
```

Some illegal variable names are:

```
1_CANNOT_START_A_VARIABLE_NAME
9CANNOT_START_A_VARIABLE_NAME_EITHER
CLOSE                variable names cannot be keywords
```

Dummy variable names are an exception. They are limited to 30 characters in length. Since the compiler must "remember" the dummy variable when defining the user defined function, a buffer of 30 characters is set aside for the name. This is not really a restriction, since it is common practice to use a single character to define the dummy variable. For example:

```
DEF FNCTION1(X) = SQR(X + X)
```

Defines function 'FNCTION1' with dummy variable 'X'.

3.2 Line Labels

BASIC normally requires an integer line number on every source line of the program. PC and XPC, on the other hand, only require a label on a line that program control will be transferred to. Also, the label need not be an integer, it can be any contiguous series of characters consisting of letters, numbers and underscores. Any other character terminates the label name. All statement labels must begin in column one, and statements must start in column two or beyond. Some examples are:

```
THIS_IS_A_LABEL REM THIS IS A REMARK STATEMENT WITH A LABEL
1000            REM THAT WAS THE LABEL '1000'
THIS_IS_A_LABEL_WITHOUT_A_STATEMENT
0000            REM NOTE THAT 0000 IS A LEGAL LABEL NAME
                GOTO 0000 \REM IS A VALID STATEMENT; BUT
                GOTO0000  \REM DEFINES A VARIABLE 'GOTO0000'
```

3.3 Continuation of Lines

PC and XPC allow BASIC lines (logical lines) to be split across physical line boundaries or in other words, a logical line consists of one or more physical lines. To do so, just place a backslash ('\') before the carriage return. PC and XPC treat the back slash-carriage return on continued lines simply as a blank. This means that variable names and keywords cannot be continued onto the next line since the blank is a separator. Also, the total length of the logical line must not exceed 255 characters or the error message 'LINE TOO LONG' will be generated. It should be noted that multiple spaces are ignored except inside of strings where they are significant. For example:

```
IF DELTA% <= GAMMA% THEN PRINT 'DELTA ='; DELTA% \<CR>
                    ELSE PRINT 'GAMMA ='; GAMMA% <CR>

* DEFINE RECORD I/O BUFFER

FOR I=0 TO NUMBER_ELEMENTS :
    FIELD 1, I*ELEMENT_SIZE AS G$,  \<CR>
              15 AS FIRST_NAME$(I), \<CR>
              15 AS LAST_NAME$(I),  \<CR>
              09 AS SOC_SEC_NU$(I), \<CR>
              02 AS INDEX$(I)  :    \<CR>
NEXT I <CR>
```

In the first line of the example, the IF-THEN-ELSE statement is compiled as one logical line even though it is split across two physical lines. Notice that a REMark statement after the THEN portion would

cause the ELSE statement to be ignored, since REMarks stop at the end of the LOGICAL line. The next logical line is the line that begins with an asterisk in column one. This is a REMark line and is ignored by the compiler. The last logical line consists of seven physical lines starting with the FOR statement and ending with the NEXT statement. Even though seven physical lines are involved, only three basic statements are used. If more than one statement is on a logical line, the statement must be seperated by either a colon (:) or a backslash (\). A backslash-carriage return does not act as a statement terminator.

3.4 Compiler Options

As the above example shows, any line that starts in column one with a separator is considered to be a comment line. In most cases, a comment line is ignored by PC and XPC. If the comment starts with a dollar sign ('$') in column one then the line is checked to see if it is a compiler option line. The following options are recognized:

TTL <string>

TTL sets the program title to <string>. The title may be 0 to 35 characters long. If the <string> is longer, anything past the 35th character is ignored. The title string is printed left justified on the same line as the date and compiler name.

STTL <string>

STTL sets the program sub-title to <string>. The sub-title may be 0 to 80 characters long. If the <string> is longer than 80 characters, anything past the 80th charcter is ignored. The sub-title string is printed left justified under the title line.

PAG

The PAG option causes a page eject to occur. Normally, a page eject is performed every 55 lines but by using the PAG option one can cause a premature page eject. See also Adapting to Your System.

SPC <n> [, <m>]

The SPC command causes <n> blank lines to be inserted into the listing. Optionally the <m> parameter can be specified which is a keep count. If there are less than <m> lines left on the page then instead of spacing <n> lines, a page eject is performed and processing of the space command is terminated. This is useful to prevent a block of lines from being split across a page.

LIS

LIS is used to turn on the listing option. If the list option is already on then the LIS option is ignored.

NOL

NOL is used to turn off the listing option. If the list option is already turned off then the NOL option is ignored.

LIB <file name>

The LIB option tells the compiler to start reading the source from an alternate file. The <file name> should be in the normal FLEX format, and will have the same defaults as the source file name did on the command line. A file that is being 'LIBed' in cannot have a LIB option in it. That is, LIB options cannot be nested or recursive.

SCALE <n>

The SCALE command is available only in XPC. It sets the SCALE factor to <n> where 0 <= n <= 6. The SCALE option must be the first line of the original source file. If an error occurs in the SCALE option the SCALE factor is set to zero. See the EXTENDED BASIC User's manual for use of the SCALE factor.

Some examples using options are:

```
$ SCALE 3     set scale factor to 3, on the first line
$ TTL this is the title up to 35 chr long
$ STTL this is the sub title, up to 80 characters long
$ PAG         perform a page eject
$ SPC 3       skip 3 lines
$ NOL         turn listing off for FILE1
$ LIB FILE1   read FILE1
$ LIS         Turn listing back on for FILE2
$ LIB FILE2   read FILE2
  END
```

4. Error Messages

There are two types of error messages that can be generated by the compilers. The first type is from errors found on the command line calling PC or XPC from FLEX. The errors are:

ILLEGAL OPTION SPECIFIED
: An unknown option character was found after the plus sign.

ILLEGAL FILE NAME
: A file name was specified that was not in the standard FLEX format.

MEMORY OVERFLOW
: Not enough memory was available to insert a symbol into the symbol table area. This can mean that the program has too many symbols to compile or that the MEMEND value in FLEX was set too low. Each program label requires the number of characters in the label plus 5 bytes for flags. Each variable requires the number of characters in the variable name plus 1 byte plus 8 bytes for double precision, 4 bytes for single precision, or 4 bytes for strings, arrays, and user defined functions.

The second type of errors are source code errors.

UNBALANCED PARENS
: An expression has unbalanced parentheses.

UNRECOGNIZABLE CHARACTER
: A character was seen that has no meaning to the compilers.

MISSING QUOTE
: A closing quote was missing from a string constant. Since strings cannot cross physical line boundaries, if an end of line is seen before a closing quote this error is generated.

DUPLICATE LINE LABEL
: Two lines have the same label.

UNDEFINED LINE LABEL
: A reference to a label that does not exist was made.

BAD CONSTANT
: An error was detected when trying to convert an ASCII number to binary. The number could be too large, too small, or contain an illegal character.

DUMMY VARIABLE NAME TOO LONG
: Dummy variable names can be a maximum of 30 characters long.

ILLEGAL SCALE FACTOR
: This error is in XPC only. The SCALE factor was too large, too

small, or the SCALE option was not on the first line of the original source file.

NESTED 'LIB' FILES NOT ALLOWED
A file being read in via 'LIB' had a 'LIB' option in it.

LINE TOO LONG
One of two things can cause this error. If the error occurred and went immediately back to FLEX then the input source was longer than 255 characters (including continued lines). If the error message was printed and the compiler continued on to the next line, then the binary generated for that line was greater than 255 bytes.

5. Adapting to your system

Since the precompilers run under the FLEX operating system, there is very little adapting to be done. None the less, there are three constants that the user can change;

(1) the page eject string that is sent to a printer,
(2) the number of lines on a page and,
(3) the number of blank lines at the top of a page.

The page eject string is sent to the printer to position the print head to the top of the next page. The eject string consists of a carriage return, line feed, form feed, and three null characters terminated by an ETX ($0004) character. The user may change the first six characters to anything else as long as the string is no more than six characters long and is terminated by the ETX character. One can disable the page eject string completely by placing an ETX as the first character of the string.

The page size is the number of lines to be printed on a page before a page eject is to be performed. Initially, this is set to 55 lines. That is, the page is assumed to be 55 lines long PLUS the number of lines defined by the margin count PLUS five lines for the page heading. The margin count is initially set to 3. This means, after the page eject is performed three blank lines are printed. Then the title and sub-title lines are printed followed by three blank lines. Then 55 lines of source followed by a page eject and so on and so forth.

It should be noted that printing the page eject string is suppressed if output is going to the terminal. So if you are listing a program at a terminal that understands form feeds, the page eject won't clear the screen on every new page.

The precompilers use as end of memory the address MEMEND defined in FLEX. As the precompilers read in source, labels and variable names are put in a symbol table. The symbol table starts at the end of the program and "grows" towards the end of memory. As a consequence, any other program to be kept in memory with the precompilers must be higher (read greater) than the address stored at MEMEND. The address of MEMEND in FLEX 1.0 and FLEX 2.0 is $AC2B, and MEMEND in FLEX 9.0 is $CC2B.

For all versions of the precompiler the constants are located at:

EJECT STRING	$0002
PAGE SIZE	$0009
MARGIN COUNT	$000A

To change any of these constants do the following:

load the precompiler into memory with FLEX's GET command,
go into your systems monitor using FLEX's MON command,

change the constant(s),
go back into FLEX at the warm start entry point
($AD03 for FLEX 1.0 and FLEX 2.0,
$CD03 for FLEX 9.0),
save the precompiler back out on the disk using FLEX's SAVE command,
where the syntax is:
SAVE <file name> <start address> <end address> <transfer address>
and you have a new version of the precompiler.

The following table defines the start, end and transfer addresses for each precompiler.

precompiler	start address	end address	transfer address
PC for 6800	$0002	$12FF	$0100
XPC for 6800	$0002	$16FF	$0100
PC for 6809	$0000	$12FF	$0000
XPC for 6809	$0000	$16FF	$0000

Appendix

The following is a list of all keywords defined in PC.

ABS	GOSUB	
AND	GOTO	PTR
AS	HEX	PUT
ASC	IF	READ
ATN	INPUT	RECORD
CHAIN	INT	REM
CHR$	KILL	RENAME
CLOSE	LEFT$	RESTORE
COS	LEN	RESUME
CVT$F	LET	RETURN
CVTF$	LINE	RIGHT$
DATA	LOG	RND
DEF	LSET	RSET
DIM	MID$	SGN
ELSE	NEW	SIN
END	NEXT	SPC
ERL	NOT	SQR
ERR	OLD	STEP
ERROR	ON	STOP
EXEC	OPEN	STR$
EXP	OR	TAB
FIELD	PEEK	TAN
FN	PI	THEN
FOR	POKE	TO
FRE	POS	USR
GET	PRINT	VAL

The following is a list of all keywords defined in XPC.

ABS	FOR	POS
AND	FRE	PRINT
AS	GET	PTR
ASC	GOSUB	PUT
ATN	GOTO	READ
CHAIN	HEX	RECORD
CHR$	IF	REM
CLOSE	INCH$	RENAME
COS	INPUT	RESTORE
CVT$2	INSTR	RESUME
CVT$F	INT	RETURN
CVT2$	KILL	RIGHT$
CVTF$	LEFT$	RND
DATA	LEN	RSET
DATE$	LET	SGN
DEF	LINE	SIN
DIGITS	LOG	SPC
DIM	LSET	SQR
DPEEK	MID$	STEP
DPOKE	NEW	STOP
ELSE	NEXT	STR$
END	NOT	SWAP
ERL	OLD	TAB
ERR	ON	TAN
ERROR	OPEN	THEN
EXEC	OR	TO
EXP	PEEK	USING
FIELD	PI	USR
FN	POKE	VAL