PED Primer

Contents

1.0 Introduction

PED, the Purdue Editor, is a text-editing system which can be used as both a batch job processor and as an interactive system with input via remote terminals. PED contains facilities to retrieve and save text information in permanent files and to manipulate local files which contain text. Text may be edited by reference to either the line numbers of the text or to the contents of text lines.

A complete description of the PED system is available in PUCC document LO-PED. This "PED Primer" is merely an introduction to PED and omits many commands and options in an effort to make the document readable in one sitting, but still give some idea of the power of the system.

2.0 Getting on and off the PED System

PED may be executed as a subsystem of PROCSY, the Purdue Remote On-Line Computing SYstem. Document LO-PROCSY is an introduction to PROCSY and includes a discussion of how to log on to PROCSY from a remote terminal and how to enter the PED system.

To enter PED from the PIRATE system, one simply types

```
+++PED
```

Upon entry, PED will respond with the log-on time and date, possibly some news messages, and an asterisk. The asterisk is a signal from PED which indicates that it is ready for a command from the user. An asterisk is issued upon completion of every command.

To end a PED session, the command TERMINATE should be used as follows:

```
*TERMINATE
```

PED will check to make sure the user has saved his files. If he hasn't, PED will ask the user if he is sure he wants to terminate. The user may then type YES or NO as appropriate.

Command names may be abbreviated to their initial letters as described in the appendix to this Primer. In the case of the TERMINATE command, the letter T is enough to uniquely define the intended operation. Therefore, the commands

```
*T
*TERM
*TERMINATE
```

are equivalent.

3.0 Permanent File Operations

The typical PED session begins by retrieving the users current text file from permanent files and ends by saving the revised text file in permanent files.

The general form of the PED command to manipulate permanent files is:

```
FILE parameters
```

where "parameters" are keywords and options which are processed as on the PFILES control card which is described in document QO-PFILES. Imbedded blanks are not allowed within PED commands.

Some examples of using permanent files from PED are:

```
*FILE(GET,MYFILE)
```
The file MYFILE is copied from the user's library and made a local file. The file is left positioned at its end.

```
*FILE(REW,TEMP)
*FILE(PUT,ABC,W=CO2A,X=TEMP)
```
The local file TEMP is rewound (by the FILE(REW,...)) and then a copy of it is saved in the user's library as file name ABC with write key CO2A.

```
*FILE,DELETE,FILE3,A=12345,N=SMITH.
```
The permanent file FILE3 is deleted from the library with user name SMITH and account number 12345.

FILE commands must be terminated by either a right parenthesis or a period. Otherwise PED will type the message ILLEGAL TERMINATION CHARACTER with an up arrow pointing to the offending character.

Files are stored in libraries associated with account numbers and user names. There are restrictions on how much can be stored in each such library. These restrictions and other valuable information about the permanent file system are discussed in document QO-PFILES.

4.0 The READ Command

Although any number of local files may be used for input and output during a PED session, PED's text-editing capabilities may be applied to only one file. This file is called the internal file and it is initially empty. The READ command is used to copy the contents of a local file to the internal file. By READing several local files, the contents of those files may be concatenated in the internal file.

The simplest form of the READ command is:

```
        READ,file
```

where "file" is the name of a local file. This command instructs PED to read the contents of one logical record of the named file and add the lines obtained to any existing text in the internal file. The named file will automatically be rewound. PED will assign line number 1.0000 to the first line in the internal file and increment the others by 1.0000. The maximum allowable number of characters per line is 140. The maximum allowable content of the internal file is approximately 13,000 50-character lines.

An example of using the READ command is:

```
        *READ,MYFILE
```

The above command instructs PED to rewind and then read the contents of one logical record of the local file MYFILE and add the lines obtained to the existing internal text.

Hence, the typical PED session will begin by using the following type of sequence of commands:

```
*FILE(GET,MYTEXT)
*READ,MYTEXT
```

5.0 Text Output Commands

The text in a users internal file may be displayed at his terminal and written to a local file by using the PRINT and WRITE commands respectively.

5.1 Displaying File Contents

The PED command to display every line of text in the users internal file at his terminal is:

```
*PRINT
```

Each line is preceded by its line number and an equals sign. To avoid the line numbers and equal signs, the word LIST should be used in place of PRINT.

To display a block of text lines rather than the entire file, address bounds may be specified as a prefix to the command. For example,

```
*150,210PRINT
```

displays only the text between lines 150 and 210 inclusive.

```
*260PRINT
```

displays only line 260.

A "CTRL-B" may be typed at any time to terminate the display of text lines. Output will not stop immediately, but should stop before three more lines are printed. Sometimes a "CTRL-B" must be repeated for it to take.

5.2 Copying to a Local File

The contents of PED's internal file may be copied to a local file by using the WRITE command. The simplest form of this command is

```
WRITE,file
```

where "file" is a name for a local file. This command instructs PED to rewind the specified file and to copy the current internal file to it as one logical record. The file is then left positioned at its end-of-record.

A typical example of using the WRITE command to save the internal file in permanent files is:

```
*WRITE,MYFILE
*FILE(REW,MYFILE)
*FILE(PUT,MYFILE)
```

If you don't rewind a file before saving it, an empty file will be stored and no warning message will be issued.

6.0 Line-Oriented Editing Commands

This section is concerned with those PED commands which edit the text in the internal file by taking an entire line as their smallest unit. Commands of this type include those which insert, delete, replace, copy and move blocks of lines.

6.1 Line Insertion

A general form of the PED command to insert one or more lines in the internal file is:

```
line INSERT increment
```

where "line" is the line number after which the insertion is to take place and "increment" is the amount by which consecutive inserted line numbers should be displaced (if possible).

An example of using the INSERT command is:

```
*5INSERT0.1
     5.1000=first inserted line is typed here
     5.2000=second inserted line is typed here
     5.3000=$
*
```

The above example causes two lines (with numbers 5.1 and 5.2) to be inserted in the internal file. PED automatically types the new line numbers and an equals sign. The user signals the completion of the insertion by typing the character $.

If an increment is not specified, PED selects an appropriate one. Whether or not an increment is specified, PED may decrease its value as the INSERT progresses in order to prevent a new line from having a line number greater than the one which originally followed the "line" of the INSERT.

One way of creating an internal file from scratch is to type INSERT without a "line" or an "increment". PED will supply line numbers to the user who can then type in his information as required. However, the most efficient method of creating a large internal file is to punch it on cards, store it in PFILES and then access it by the FILE command.

6.2 Line Deletion

A general form of the PED command to delete one or more lines from the internal file is:

```
lines DELETE
```

where "lines" is required and is either a single line number or two line numbers separated by a comma. The single line number indicates only one line is to be deleted and the pair of line numbers indicate that all lines inclusive between the two are to be deleted.

Some examples of deleting lines are:

```
*18DELETE
     delete line number 18.

*45,73DELETE
     delete all lines between 45 and 73 inclusive.
```

The most efficient way to delete <u>all</u> lines in the internal file is to use the command PURGE as follows:

```
*PURGE
```

6.3 Line Replacement

A general form of the PED command to replace one or more lines in the internal file is:

```
     lines REPLACE
```

The REPLACE command is a combination of the DELETE and INSERT commands. Each "line" specified is first DELETEd and then PED accepts a new line from the user as with the INSERT command.

An example of using the REPLACE command on a file with line numbers 4, 5, 5.1, 5.2, 6 and 7 is:

```
*5,6REPLACE
   5.0000=first replacement line
   5.1000=second replacement line
   5.2000=third replacement line
   6.0000=fourth replacement line
*
```

PED automatically types the line number and equals sign to prompt the user. The above example causes lines 5, 5.1, 5.2 and 6 to be replaced.

The replacement of lines may be terminated before the exhaustion of line numbers in the specified bounds by typing $ as in the following example which again assumes line numbers 4, 5, 5.1, 5.2, 6 and 7 are present:

```
*5,6REPLACE
   5.0000=first replacement line
   5.1000=second replacement line
   5.2000=$
*
```

The preceding example causes lines 5 and 5.1 to be replaced, but leaves lines 5.2 and 6 untouched.

The replacement of only one line is signaled by using only one line number in the REPLACE command as follows:

```
*5REPLACE
   5.0000=replacement line
*
```

6.4 Line Copying and Moving

A general form of the PED command to copy a block of lines is:

```
lines COPY number
```

where "lines" delimits the block to be copied and "number" is the line number after which the copy is to be inserted. The line numbers and text within "lines" are not modified. "number" may not lie within "lines".

PED automatically determines the line numbers to be used for the copied lines. There must be a sufficient number of unused line numbers between "number" and the following line to allow all of the copied lines to be inserted. No line is copied unless all can be copied. For example it would not be possible to copy a block of five lines to those lines between line numbers 2.995 and 3.000 inclusive.

For the following example, assume that there are 100 lines in the internal file, numbered from 1.0 to 100.0 in increments of 1.0. Then the PED command

```
*20,25COPY80
```

would create six new lines with numbers 80.1 through 80.6 in increments of 0.1. These new lines would contain the text which was in the original lines numbered 20 through 25. Those original lines would still be present in the internal file after the COPY was completed.

If the word MOVE is used in place of the word COPY, PED will perform the copy and then destroy the original lines. Thus the MOVE command causes a block of lines to be transferred.

7.0 String-Oriented Editing Commands

In addition to its ability to edit entire lines of text, PED can locate and manipulate strings of characters within lines. The simplest form of a PED command which does this is:

lines EDIT,string$_1$==string$_2$$

where "lines" specifies the line numbers within which the editing is to take place and the two "strings" are strings of characters delimited by single quote marks. PED will search for every occurrence of "string$_1$" and replace it by "string$_2$". The dollar sign character is required after "string$_2$".

For example, the command

```
*1,900EDIT,'SINF('=='SIN('$
```

instructs PED to search between line numbers 1 and 900 inclusive for any occurrence of the characters "SINF(". When one is found, the characters "SIN(" are substituted in its place and the search continues. Since the strings are of unequal length, PED automatically adjusts the positions of the remaining characters of each affected line.

Some of the possible variations on this form of editing are as follows:

- leave off the "lines"
  By default the entire internal text file is processed.
- leave off one of the equals signs
  This causes only the first occurrence of the specified characters to be replaced.
- place an integer number between the two equals signs
  This causes only the first "integer" occurrences of the specified characters to be replaced.

For example, an efficient way of correcting line number 83.4 which contains the characters:

```
THIS IS A LMNE
```

would be to use the PED command

```
*83.4EDIT,'LMNE'='LINE'$
```

There are several other forms of the EDIT command and several other PED commands which modify text based on line content. The interested user is invited to read the appropriate sections of the PED Reference Manual (LO-PED) for more information.

8.0 More about PED

As indicated at the end of the last section and in the Introduction, there is quite a bit more to PED than is described in this Primer. The complete documentation of the entire PED system is in the PED reference manual. At this point, however, it is worth glancing at the appendix to this Primer to get a feel for the type of commands that are available. There are two remarks worth making after a quick look at that appendix. First of all, there are many more PED commands than those which have been covered in the preceeding chapters. We leave the description of those commands to the PED reference manual. Secondly, there is more to the commands covered than was included in their descriptions in this Primer. This chapter addresses itself to explaining some of those options on PED commands. Once again a complete listing of all available options is included in the PED reference manual.

8.1 Address Bounds

Most PED commands are optionally preceded by address bounds. If only one line address is specified in the bounds, then the command applies to only one line. If two line addresses are given separated by a comma, then the command applies to all lines inclusively between the two specified. If address bounds are not specified, the command applies to all text lines. (A notable exception is the DELETE command which requires address bounds.)

The normal way of specifying a line address is to give its line number using digits and possibly a decimal point. Certain special characters are also allowed in defining a line address. These characters and their meanings are as follows:

↑	The first line of text.
!	The last line of text.
=	All the lines referenced by the preceding command.
=↑	The first line referenced by the preceding command.
=!	The last line referenced by the preceding command.

Some examples of using these special characters in line addresses are:

```
*=PRINT
```
Print all lines referenced by the preceding command.

```
*38,!DELETE
```
Delete all lines from 38 to the end of the file.

```
*↑,=!WRITE,TEMP
```
Copy from the first line of the internal file to the last line referenced in the previous command and make it a local file named TEMP.

An alternate means of specifying a line address is by line context. Let us define a literal as a string of characters delimited by single quotes and then the following forms are allowed as line addresses by context:

literal	The first line with the string anywhere in that line.
literal column number	The first line with the string beginning at the specified column of that line.
$[\text{column}_1,\text{column}_2]$literal	The first line with the string appearing within the specified column bounds.

Some examples of using line context to specify addresses are:

*'PROGRAM'REPLACE
Replace the first line which contains the literal "PROGRAM" anywhere within the line.

*21,'END'7LIST
List all lines from line 21 to the first which contains the characters "END" in columns 7, 8 and 9.

*[50,72]'SUB2'MOVE230
Move the first line which contains the characters "SUB2" between columns 50 and 72 to the line after line number 230.

The specification of a line address by context may be combined with other contexts by the use of the following logical operators:

\	(reverse slash)	NOT
&	(ampersand)	AND
"	(double quote)	OR

Normally, the NOT is applied before the AND, and the AND is applied before the OR. However, parentheses may be used to group operations and force a different order.

Some examples of using combined contexts to specify addresses are:

*\('PRINT'"'READ')PRINT
Print the first line that contains neither the characters "PRINT" not the characters "READ".

*'ALPHA'2&'BETA'DELETE
Delete the first line that contains "ALPHA" in columns 2 through 6 and "BETA" anywhere within the line.

8.2 Environment Control

PED operates under the restrictions of several user-controllable "environment variables." These variables may be set or reset during a PED session by using the ENVIRONMENT command as follows:

```
ENVIRONMENT(variable=value,...,variable=value)
```

where the most commonly-used "variables" and their possible "values" are listed and described in the following paragraphs.

Variable	Name and Default	Possible Values
- String quote	QU='	any single symbol
- Rewind control flag	RE=1	unsigned integer
- Line length	LE=72	unsigned integer≤140
- Entry terminator	TE=$	any single symbol
- I/O mode	IO=B	B or C (Binary or Coded)
- Line terminator	LT=undefined	any single symbol

- String quote — The string quote symbol is used to delimit strings.
- Rewind control flag — If zero, local files are not automatically rewound before READ and WRITE. If non-zero, they are automatically rewound.
- Line length — Controls the length of lines used in READ and WRITE commands. Only "LE" characters/line are read and written.
- Entry terminator — This symbol signals the end of user input for those commands which allow more than one line per use such as INSERT, REPLACE and EDIT.
- I/O mode — Specifies the mode of READ and WRITE operations.
- Line terminator — Used as a separator character for lines containing more than one command. (However if subsequent input is expected by such commands, it must also appear on the input line.)

It is possible to set or reset an environment variable value to "undefined" by equating it to any number greater than 63. An example of using the ENVIRONMENT command is

```
*ENVIRONMENT(TE=;,LT=64)
```

which makes the semi-colon the entry terminator and sets the line terminator to undefined. Note that normally the line terminator character is undefined so that only one PED command may be entered per line.

8.3 Options

Most PED commands allow "options" to be specified after the command name. Several types of options are available.

Any combination of the following letters may be typed after the command name to select the "option" as described:

- A
  The "A" option means print each line <u>after</u> processing it.

- B
  The "B" option means print each line <u>before</u> processing it.

- W
  The "W" option means print each line <u>before</u> processing it and <u>wait</u> for permission to process it. PED will type the message

```
                    OK?
```

after typing the line and will wait. If the user types "NO", PED will not modify the line but will continue execution of the command. If the user types "YES" or "Y" or nothing (i.e. just a carriage return), PED will make the line modification as usual and then continue execution of the command.

Some examples of using these options are:

```
*8,12DELETEW
   Print each line between 8 and 12 and wait for permission to delete it.

*EDITBA,'ABCD'=='WXYZ'$
   Find and print each line which contains the characters "ABCD". Change those
   characters to "WXYZ" and then print the revised line.
```

A second type of PED option allows conditions on the command to be specified in the form of line contexts. The format of line contexts and their meanings were discussed in the section on address bounds so we will just illustrate their use as options by examples in this section:

```
*PRINT'SUBROUTINE'
   Print every line which contains the characters "SUBROUTINE".

*↑,!DELETE'DEBUG''''COMMENT'
   Delete every line which contains either the characters "DEBUG" or the characters
   "COMMENT".
```

A third type of PED option allows an "environment variable" to be reset for one command only. Temporary environment specifications are enclosed in parentheses as shown in the following examples:

```
*READ(RE=0),FILE3
   Read local file FILE3 without rewinding it.

*WRITEB'A''''B'(LE=20),TEMP
   Copy the internal file to a local file named TEMP, but only transfer the first
   20 columns of those lines which contain either the character "A" or the character
   "B". Also print each line before it is written to TEMP.
```

9.0 Batch PED

PED may be executed as a batch job. In the description below of the PED control card syntax, the square brackets ([,]) indicate optional parameters. Parameters may appear in any order on the control card.

```
PED(cfile[,ofile][,F=fname][,evn=value,...,evn=value])
```

where "cfile" - the name of the file containing PED commands. No file positioning is done by PED before opening "cfile". Default "cfile": none.

"ofile" - the name of the file to contain PED output. No file positioning is done by PED before opening "ofile". Default "ofile": OUTPUT.

"fname" - if the "F=name" parameter is included on the control card, file "fname" is read into the text file prior to executing any commands from "cfile". File "fname" is rewound before the read unless the RE environment variable has been set to zero on the control card. If a Terminate command is encountered on "cfile" before a Write command, PED will write the contents of any text file in expanded form onto file "fname" and then terminate. If <u>any</u> Write command is encountered before a Terminate command is encountered, PED will not write onto file "fname" before terminating. Default "fname": none.

"env" - the name of any valid environment variable as described in section 8.2 of this document. Default "env": none.

"value" - the value to which the selected permanent environment variable is to be set prior to execution of any commands on "cfile". "Value" is assumed to be a decimal integer unless post-fixed by a B in which case it is taken to be on octal integer.

Some examples of the PED control card are given below.

(1) PED(INPUT)

Commands are read from file INPUT and PED output is placed on file OUTPUT.

(2) PED(INPUT,F=TEXT)

File TEXT is rewound and read into the PED internal text file. Commands are then read from file INPUT, and PED output is placed on file OUTPUT.

(3) PED(COMND,OUTFILE,F=TEXT,RE=0,LT=50B)

File TEXT is read from its current position into the PED internal text file since the rewind flag has been reset to zero. The line terminator character is set to the value 50 octal corresponding to a display code slash (/). Commands are read from file COMND and PED output is written on file OUTFILE.

Appendix - Instant PED

This appendix is a summary of all available PED commands. The commands are listed alphabetically and contain a one-line description, command syntax, and special options if any. Only certain initial characters of command names are significant. These characters are typed in upper case in the syntax. Any member of a class of items is indicated by enclosure in angular brackets <...>.

AGAIN

Repeat the immediately preceding command.

<address bounds> Again

APPEND

Add text to the end of internal text file with line number increment of 1.0 until user types the character $.

APpend<options>

BROWSE

Display fifty lines of text on the screen of the 252 graphics terminal.

Browse
  or
<lower bound>
  or
carriage return

CHANGE

Create or change text within specified fields until user types the character $.

<address bounds> Change <options>, <first change>

special options: L left-adjusted changes
R right-adjusted changes

COLUMNS

Display the line number and starting and ending column numbers of each occurrence of "context".

<address bounds>COLumns<options>,<context>

COPY

Copy a block of lines to a specified line.

<address bounds>COpy<options><line number>

CPTIME

Display the remaining central processor time in seconds.

```
CPtime
```

DELETE

Delete a block of lines.

```
<address bounds>Delete<options>
```

EDIT

Replace portions of "context".

```
<address bounds>Edit<options>,<context><count><replacement>$
```

special option: P Paired searches

ENVIRONMENT

Reset an environment variable.

```
ENvironment(<name>=< value>,...,<name>=<value>)
```

EXECUTE

Execute a MESA processor.

```
EXecute,<MESA control card>
```

FILE

Manipulate a local or permanent file.

```
FILe,<function>,<file information>
```

```
functions:  EOF  Write end-of-file.                SNF   Search for named file.
            EOR  Write end-of-record.              SNR   Search for named record.
            REL  Release.
            RET  Return.                        DELETE PFILES delete.
            REW  Rewind.                        GET    PFILES get.
            SEI  Skip to end-of-information     INDEX  PFILES index.
            SFB  Skip file backward.            PUT    PFILES put.
            SFF  Skip file forward.
            SRB  Skip record backward.
            SRF  Skip record forward.
            UNL  Unload.
```

FIND

Add to or delete from the FIND list.

```
<address bounds>FINd<options>
```

special options: + Add to FIND list
- Delete from FIND list

INSERT
Insert one or more lines until user types $.

```
<line address> Insert<options><increment>,<first line>
```

JUSTIFY
Reposition character strings within a line and eliminate unwanted leading or trailing characters in the strings.

```
<address bounds>Justify <options>,<column 1>,<column 2>,<column3>,<column4>
```

```
Special Options:  L  Left justification
                  R  Right justification
                  D = <char1> Delete<char1>
                  D = (<char2>,<char3>) Delete <char2> on left end and <char3>
                     on right end
                  F = <char-4> Fill with <char-4>
```

LABEL
Display the number and first non-blank characters for each line with a non-blank character in column one.

```
<address bounds>Label<options>
```

LENGTH
Display the length of lines.

```
<address bounds>LEngth<options>
```

LIST
Display the text of lines without line numbers.

```
<address bounds>LIst<options>
```

MOVE
Transfer a block of lines.

```
<address bounds>MOve<options><line number>
```

NUMBERS
Display the numbers of lines.

```
<address bounds>Numbers<options>
```

PRINT
Display the numbers and text of lines.

```
<address bounds>Print<options>
```

PURGE
Delete all lines.

```
PUrge<options>
```

READ
Read text from local file.

```
Read<options>,<file name>
```

Special Options:	O	Use old line numbers
	N	Use new line numbers

RENUMBER
Renumber lines.

```
<address bounds>RENumber<options>,<starting number>,<increment>
```

REPLACE
Replace a block of lines.

```
<address bounds>REPlace<options><increment>,<first line>
```

SHIFT
Move characters from <column-1> through <column-2> inclusive into <column-3> through <column-4> inclusive.

```
<address bounds>Shift<options>,<column-1>,<column-2>,<column-3>,<column-4>
```

TAB
Set or reset tab stops.

```
TAb,<integer>,...,<integer>
```

TERMINATE
Terminate the PED session.

```
Terminate
```

WRITE
Write text into a local file.

```
<address bounds>Write<options>,<file name>
```

Special Options:	O	Use old line numbers
	N	Use new line numbers
	C	Use compressed text format