PICK®

SYSTEMS

THE PICK® SYSTEM
R83 USER
REFERENCE MANUAL

VOLUME 2

The PICK System
R83 User Reference Manual

Printed in the United States of America.

✓ PPG 4
4/17/88

Chapter 8

RUNOFF

THE PICK SYSTEM

USER MANUAL

Contents

8 RUNOFF

8.1 RUNOFF INTRODUCTION AND RUNOFF VERB FORMAT 8-3
8.2 RUNOFF SOURCE FILE FORMAT 8-4
8.3 RUNOFF COMMANDS 8-5
8.3.1 BEGIN PAGE (BP) 8-5
8.3.2 BOX n,m / BOX OFF (BOX) 8-5
8.3.3 BREAK (B) 8-5
8.3.4 CAPITALIZE SENTENCES (CS) 8-5
8.3.5 CENTER (C) 8-6
8.3.6 CHAIN {[DICT] [FILE-NAME]} ITEM-ID 8-6
8.3.7 CHAPTER text 8-7
8.3.8 COMMENT INSTRUCTION (*) 8-7
8.3.9 CONTENTS 8-7
8.3.10 CRT 8-7
8.3.11 FILL (F) 8-7
8.3.12 FOOTING 8-8
8.3.13 HEADING 8-8
8.3.14 HILITE c / HILITE OFF 8-9
8.3.15 HYPHENS 8-9
8.3.16 INDENT n (I) 8-9
8.3.17 INDENT MARGIN n (IM) 8-9
8.3.18 INDEX text 8-10
8.3.19 INPUT 8-10
8.3.20 JUSTIFY (J) 8-10
8.3.21 LEFT MARGIN n 8-10
8.3.22 LINE LENGTH n 8-10
8.3.23 LOWER CASE (LC) 8-10
8.3.24 LPTR 8-10
8.3.25 NOCAPITALIZE SENTENCES (NCS) 8-11
8.3.26 NOFILL (NF) 8-11
8.3.27 NOJUSTIFY (NJ) 8-11
8.3.28 NOPAGING (N) 8-11
8.3.29 noparagraph 8-11
8.3.30 page number n 8-11
8.3.31 PAPER LENGTH n 8-11
8.3.32 PARAGRAPH n 8-11
8.3.33 PRINT INDEX 8-13
8.3.34 PRINT 8-13
8.3.35 READ {[DICT] [file-name]} item-id 8-13
8.3.36 READNEXT 8-13
8.3.37 SAVE INDEX file-name 8-17
8.3.38 SECTION n text 8-17
8.3.39 SET TABS n,n,n, ... 8-17
8.3.40 SKIP n (SK) 8-18
8.3.41 SPACE n (SP) 8-18
8.3.42 SPACING n 8-18
8.3.43 STANDARD 8-18
8.3.44 TEST PAGE n 8-18
8.3.45 UPPER CASE (UC) 8-18
8.4 SPECIAL CONTROL CHARACTERS 8-19
8.4.1 Upper and Lower Case Controls 8-19
8.4.2 Underlining and Overstriking 8-20
8.4.3 Tab Settings 8-21

8.1 RUNOFF INTRODUCTION AND RUNOFF VERB FORMAT

RUNOFF is a verb which facilitates the preparation and maintenance of textual material such as memos, manuals, etc. RUNOFF takes text prepared with the EDITOR and produces formatted output. RUNOFF source text contains commands which control justification, page titling and numbering, spacing and capitalization. Textual material prepared with RUNOFF may be easily edited and corrected with the Editor and then reprinted with RUNOFF. Material may be inserted or deleted, while unchanged text need not be retyped. RUNOFF also provides the capability of combining separate textual material into a single report and inserting duplicate text into different reports.

RUNOFF is the TCL-II verb issued to process one or more source text file items in RUNOFF format. Multiple input items are treated as a single source text file. A source text item may contain a command which causes RUNOFF to CHAIN to another file item. This makes it possible to CHAIN file items together without doing a SELECT or SSELECT. Items included in the RUNOFF verb's item-list may chain to other items within the same file. when the chain ends, processing continues with the next item from the item-list.

A source text item may also contain a command which causesRUNOFF to READ a second file item and then resume processing of the first item. This makes it possible to insert the text from a single file item in the output from many other file items (see example below).

The RUNOFF verb format is: RUNOFF file-name item-list {(options)}

OPTIONS:

C The C option suppresses thecommands.

I The I option will output the name of the next item to be 'Runoff'. (helpful for tracing.CHAINed sequences)

J The J option will suppress Highlighting.

N The N option causes output to the terminal to be continuous; that is, RUNOFF will not pause at the bottom of a page and wait for a carriage-return if the N option is used.

Nnn This numeric option may be used to set the number of times BOLDFACE letters are overprinted.

P The P option may be used to direct output to the line printer.

S The S option may be used to suppress underlining and boldface when RUNOFF output is directed to a CRT.

U The U option will force the output to upper-case.

8.2 RUNOFF SOURCE FILE FORMAT

The source file contains the textual material which will appear on the final copy, plus command information to specify formatting and alternate sources of input.

Each line of input source text is processed in the text mode except those beginning with a period. A line beginning with a period is assumed to be a command line and is processed in the command mode. A command line may contain one or more commands, each starting with a period. The commands provide formatting information and select various modes of
RUNOFF fills each output line by adding successive words from the source text until one more word will not fit on the line. The line is then justified by inserting blank spaces between words at random until the last word in the line exactly meets the right margin. RUNOFF may be set to fill output lines without justifying the right margin. When filling lines, spaces and end-of-lines are treated only as word separators. Multiple word separators are stripped from the input. RUNOFF may be set to transmit the input source text to the output without filling lines or justifying margins. In this mode, multiple spaces and end-of-lines are not stripped from the input. Some of the commands cause a BREAK in the output. A BREAK means that the current line is output without justification. This occurs at the end of paragraphs.

```
.SK.xbox 1,78.SK

.BP.F.J.PARAGRAPH 0.LEFT MARGIN 2.LINE LENGTH 74

.SECTION 1 INTRODUCTION TO RUNOFF

.INDEX 'RUNOFF Introduction'

.box OFF.C
```

Common RUNOFF Commands.

8.3 RUNOFF COMMANDS

RUNOFF commands are stored along with the textual material in the source file. These commands are distinguished by a period at the start of a command line. A command line may contain one or more command, each starting with a period. The commands provide formatting information and select various modes of operation.

Note: In the following descriptions of RUNOFF commands, valid command abbreviations are
enclosed in parantheses (where such abbreviated forms of the command exist).

8.3.1 BEGIN PAGE (BP)

BEGIN PAGE causes a BREAK (see below) followed by an advance to a new page. The page number is incremented and the page heading (if set) is printed.

8.3.2 BOX n,m / BOX OFF (BOX)

The BOX command causes the following text to be enclosed in a box with the width parameter specified by 'n' (right margin) and 'm' (left margin). The text will continue to be 'boxed' until a "BOX OFF" command is encountered.

For example:

```
001 .box 4,74.CENTER
002 This is an example of a BOX.
003 .box
```

This is an example of a BOX.

8.3.3 BREAK (B)

BREAK causes any partially filled line to be output before processing the next input line.

8.3.4 CAPITALIZE SENTENCES (CS)

This command puts RUNOFF in the capitalize sentences mode. In this mode the first letter of each sentence is capitalized. The first letter after a '.', '?', or '!', followed immediately by either a space or an end-of-line (Attribute Mark) is capitalized. The capitalize sentences mode also causes the following characters to be followed by a double space or an end-of-line: '.', '?', '!', ':', and ';'. CAPITALIZE

SENTENCES is one of the STANDARD settings. (See the STANDARD command.) note: This command is ignored in the NOFILL (NF) mode.

8.3.5 CENTER (C)

CENTER causes the next line to be input in NOFILL mode and centered on the next line of output. This command causes a BREAK to occur.

8.3.6 CHAIN ([DICT] [FILE-NAME]) ITEM-ID

This command causes RUNOFF to CHAIN to the input text file item indicated. The [DICT] and [FILE-NAME] are both optional. If DICT is not specified, the DATA section of the file is assumed. If no FILE-NAME is given, the item will be read from the same file as the item being processed.

The text input from this item is processed and output without any parameter or mode changes. RUNOFF does not resume processing text from the current source of input. This command does <u>not</u> cause a BREAK.

The .CHAIN command will scan the string following the command, looking for an item-id or a file name. The legal delimiter for the item-id or file name is a blank. They may have an included period. If there is more than one string following the CHAIN command which is delimited by a blank, then the next-to-the-last field will be taken to the the file name, and that file will be opened. The last field delimited with a blank will be considered the item-id, and it will be retrieved by the RUNOFF processor to be executed next. You can include a comment statement after the CHAIN, however. Therefore, for the purposes of the CHAIN commands, the line is considered exhausted when the processor encounters an end-of-line mark, or when it encounters a period preceeded by a space.

If the processor opens a file when executing a CHAIN statement, that file will be the file from which all succeeding items are retrieved, until the file is respecified by another CHAIN statement.

The C option will suppress the .CHAIN command if it is desired to RUNOFF one element of a chained or tree-ed structure. The I option will cause the name of the next item to be output by RUNOFF to be placed in the last line of the last item RUNOFF. This is of use with relatively large documents.

8.3.7 CHAPTER text

This command may be used to handle automatic chapter numbering and formatting. This command has the same effect as:

```
.BEGIN PAGE.CENTER
.CHAPTER n
.SPACE 2
text
.SPACE 2
```

where the chapter number n is incremented automatically. For example:

```
.CHAPTER RUNOFF
```

would produce:

CHAPTER 8

RUNOFF

8.3.8 COMMENT INSTRUCTION (*)

This command informs the RUNOFF processor that all of the rest of the text in the line in which it occurs is a comment. It must either be at the beginning of the line, or after another command in a command line. It is always the last command in a line. This allows text to be commented out, and the intent of READs and CHAINs to be noted.

8.3.9 CONTENTS

This command prints the table of contents accumulated by preceding CHAPTER and SECTION commands. This command should be used at the end of the RUNOFF source file. an example of the results of this command can be seen by looking at the TABLE OF CONTENTS at the beginning of this manual. Note: the LINE LENGTH and LEFT MARGIN of the Table of Contents is determined by those settings that are in effect when the first .CHAPTER or .SECTION command is encountered.

8.3.10 CRT

This command directs the RUNOFF output to the user's terminal. CRT is one of the STANDARD settings. (See the STANDARD command.)

8.3.11 FILL (F)

FILL puts RUNOFF into the line fill mode. Words are processed until there are enough to fill a line without overflowing it. If justifaction mode is on, RUNOFF will insert spaces in the line at random to make the right margin line up. FILL is one of the STANDARD settings. (See the STANDARD command.)

8.3.12 **FOOTING**

FOOTING causes the next line to be input in nofill mode and stored in a page footing buffer. The page footing buffer will be output at the bottom of each page. The page footing may be changed with successiveFOOTING commands. The following characters have special meaning in page footings and headings:

'P' Prints out the page number, right justified in a field of four spaces, with blank fill.

'Pn' Prints out the page number, left justified in a field of 'n' spaces ('n' specified by the user).

'L' Performs a carriage return/line-feed (CR/LF).

'i' Prints out the Item-Id.

'in' Prints out the Item-Id, left justified in a field of 'n' spaces ('n' specified by the user).

'F' Prints out the File-Name.

'Fn' Prints out the File-Name, left justified in a field of 'n' spaces.

'T' Prints out the Time and Date (22 characters long).

'D' Prints out the Date in '01 JAN 1977' format (11 characters).

'C' Centers the line.

FOOTING causes a BREAK and also is one of the STANDARD settings. (See the STANDARD command.)

8.3.13 **HEADING**

HEADING causes the next line to be input in NOFILL mode and stored in a page heading buffer. The page heading buffer will be output at the top of each page.

The page heading may be changed with successive HEADING commands. The special characters described under the FOOTING command may also be used in page headings.

The HEADING command causes a BREAK and also is one of the STANDARD settings. (See the STANDARD command.)

8.3.14 HILITE c / HILITE OFF

HILITE causes the character specified by 'c' to be printed out at the extreme right margin for every line of text until a HILITE OFF command is encountered. An example of the HILITE command may be seen at the right of this text. *

The highlight command does not cause a break in the text. This allows parts of paragraphs to be highlighted in justify or fill mode. If you wish to align the HILITE command with a paragraph, it may be necessary to put the HILITE | command after the first line of filled or justified text, and to put the form .BREAK at the end of the paragraph. *

The execution of the hilite command also is such that if the term is the last character string in command line, then it is equivalent to HILITE OFF. The J option will suppress highlighting.

8.3.15 HYPHENS

Hyphens which are surrounded by alphabetic characters will allow a word-break on the hyphen in fill and justify modes. That is, if a term is a concatenation of two words separated by a hyphen, and the line overflows within the second part of the term, then the first part and the hyphen are left in the line, and the next line is commenced with the second part of the word.

Similarly, if a line in the source text terminates with a hyphen preceeded by an alphabetic character, and the first character in the next line is an alphabetic character, then the last word in the line and the hyphen will be concatenated with the first word in the next line and output together in a line with the hyphen between the two parts. If there is a line overflow which occurs during this process, the hyphenated word will be handled as above. What the processor will <u>not</u> do is remove the hyphen.

If the hyphen does not have this meaning, then the back-arrow character may be placed in front of it to suppress this action.

8.3.16 INDENT n (I)

INDENT causes the next line of output to be indented by n spaces to the right of the left margin. n may be negative to cause the line to begin left of the left margin. If n is missing, n=1 is assumed. This command causes a BREAK to occur.

8.3.17 INDENT MARGIN n (IM)

This command causes the left margin to be increased by n spaces and the line length to be decreased by n. Negative n may be used to decrease the left margin and increase the line length. This command causes a BREAK to occur.

8.3.18 INDEX text

INDEX causes the text specified to be stored in an index list. The text may be more than one word, or several words enclosed in single quotes. The word, or words, along with the current page number, are put in a sorted index list. The index can be printed by the PRINT INDEX command.

8.3.19 INPUT

The INPUT command caused RUNOFF to read the next line of source text from the user's terminal. The text input from the terminal is processed and output without a BREAK or mode change.

8.3.20 JUSTIFY (J)

JUSTIFY puts RUNOFF in the FILL and JUSTIFY mode. RUNOFF fills each output line by adding successive words from the source text until one more word will not fit on the line. the line is then justified by inserting blank spaces between words at random until the last word in the line exactly meets the right margin. JUSTIFY is one of the STANDARD settings. (See the STANDARD command.)

8.3.21 LEFT MARGIN n

This command sets the left margin to n spaces. If n plus the current line length exceeds the maximum line length, this command is ignored. A LEFT MARGIN of 0 is one of the STANDARD settings. (See the STANDARD command.)

8.3.22 LINE LENGTH n

This command sets the line length to n characters (not counting the left margin). If n plus the current left margin exceeds the maximum line length, this command is ignored. A LINE LENGTH of 70 is one of the STANDARD settings. (See the STANDARD command.)

8.3.23 LOWER CASE (LC)

This command puts RUNOFF into lower case mode. In lower case mode all letters are automatically made lower case. They may then be changed to upper case by various text commands or control characters. (See the section on RUNOFF Special Characters.)

8.3.24 LPTR

This command directs the RUNOFF output to the line printer.

8.3.25 NOCAPITALIZE SENTENCES (NCS)

This command resets the CAPITALIZE SENTENCES mode.

8.3.26 NOFILL (NF)

This command resets both the JUSTIFY and FILL modes. Input text lines will be output as they are, (after posssible elimination of special control characters) without removal of extra spaces. Output lines will not be filled nor will right margins be justified. This command causes BREAK.

8.3.27 NOJUSTIFY (NJ)

This command resets the JUSTIFY mode, but has no effect on the FILL mode.

8.3.28 NOPAGING (N)

The N option may be used to eliminate the wait for terminal input at the end of each page printed on the terminal.

8.3.29 NOPARAGRAPH

This command resets the paragraph mode. Blank input text lines and spaces at the beginning of a line will be ignored in justify mode.

8.3.30 PAGE NUMBER n

This command sets the current page number to n. If n is missing, n=1 is assumed.

8.3.31 PAPER LENGTH n

This command sets the paper length to n lines.

8.3.32 PARAGRAPH n

This command causes any blank line or any line which starts with a space to be considered as the start of a new paragraph. This allows normally typed text to be justified without any special commands. n sets the number of spaces paragraphs are to be indented or unindented. A paragraph causes a BREAK followed by (line spacing + 1)/2 blank lines. A PARAGRAPH 5 is one of the STANDARD settings. (See the STANDARD command.)

The PARAGRAPH command may be set to a negative number. The example shows the use of a negative paragraph setting to decrease the left margin.

```
001 .SK .PARAGRAPH -4 .LEFT MARGIN 13 .LINE LENGTH 63
002  1.  The user enters the command "Z" to the DEBUGGER prompt character
003 "*". The DEBUGGER responds with "PROG NAME?", the user enters the
004 program name. This allows the DEBUGGER access to the symbol table
005 created during compilation. Alternatively, if the user uses the
006 "(D)" during run time, access to the symbol table is already
007 established, and use of the "Z" command is unnecessary.
008  2.  To find out how far in the loop the program progressed, the
009 user looks at the variable "I" by entering "/I". The DEBUGGER
010 responds with
011 "11 -", at which the user may change the value of "I" if desired.
012 The user may then want to look at all of the values in the array by
013 entering "/ARRAY". The DEBUGGER responds with "ARRAY(1)-1-", the
014 user depresses
015 return and the DEBUGGER continues with the next "array slot"
016 (i.e., "ARRAY(2 etc.)-2-" ). Once "ARRAY(10)-10-" has been reached
017 the . . . etc.
018
```

Note that in the above example the text lines beginning with 1. and 2. are spaced over one space thus resulting in the negative paragraphing.

The above source text would print:

1. The user enters the command "Z" to the DEBUGGER prompt character "*". The DEBUGGER responds with "PROG NAME?", the user enters the program name. This allows the DEBUGGER access to the symbol table created during compilation. Alternatively, if the user uses the debug option "(D)" during run time, access to the symbol table is already established, and use of the "Z" command is unnecessary.

2. To find out how far in the loop the program progressed, the user looks at the variable "I" by entering "/I". The DEBUGGER responds with "11 -", at which the user may change the value of "I" if desired. The user may then want to look at all of the values in the array by entering "/ARRAY". The DEBUGGER responds with "ARRAY(1)-1-", the user depresses return and the DEBUGGER continues with the next "array slot" (i.e., "ARRAY(2)-2-" etc.). Once "ARRAY(10)-10-" has been reached the ... etc.

Sample usage of a negative PARAGRAPH command.

8.3.33 PRINT INDEX

This command causes the sorted index list of words and page numbers to be printed. The index is sorted into alphabetical order, and printed in two columns per page. Note -- this command changes the tab settings, and causes a BEGIN PAGE command to be performed.

8.3.34 PRINT

The PRINT command causes RUNOFF to print the next line of input text on the user's terminal.

8.3.35 READ ([DICT] [file-name]) item-id

This command causes RUNOFF to read the file item indicated. The [DICT] and [FILE-NAME] are both optional. If DICT is not specified, DATA section of the file will be used. If no FILE-NAME is given, the item will be read from the same file as the item being processed. The text input from this item is processed and output without any parameter or mode changes. After processing this item, RUNOFF resumes input with the next line of the current source of input. This command does not cause a BREAK.

The .READ command will scan the string following the command, looking for an item-id or a file name. The legal delimiter for the item-id or file name is a blank. They may have an included period. If there is more than one string following the READ command which is delimited by a blank, then the next-to-the-last field will be taken to the the file name, and that file will be opened. The last field delimited with a blank will be considered the item-id, and it will be retrieved by the RUNOFF processor to be executed next. If the statement is a READ, then the processor will eventually return to this item and continue processing it. When it does, it will commence at the beginning of the next line in the item. Therefore, no statements which occur after the READ statement in the line will be executed. You can include a comment statement after the READ however. Therefore, for the purposes of the READ command, the line is considered exhausted when the processor encounters an end-of-line mark, or when it encounters a period preceeded by a space. The C option will suppress the .READ command if it is desired to RUNOFF one element of a chained or treed structure. The I option will cause the name of the next item to be output by RUNOFF to be placed in the last line of the last item RUNOFF. This is most useful with large documents.

8.3.36 READNEXT

This command is used to read data from a pre-selected LIST. It has an effect only if, prior to entering RUNOFF, a SELECT, SSELECT, QSELECT or GET-LIST statement has been entered, which selects a list of values. Each READNEXT command in RUNOFF will extract one value from the select-list and place it in the text stream. READNEXT does not cause a break. If

there is no pre-selected list, or when the list is exhausted, the READNEXT command will cause a termination of RUNOFF, and a return to TCL.

This command is particularly useful when form-letters are to be generated. For example, it may be necessary to insert the name and address of each recipient of the letter from a separate file. A SSELECT statement is used to extract the relevant data from the file and save it in a list. A series of READNEXT statements will insert the data into the text of the letter. At the end of the letter, a CHAIN statement may be used to restart the next letter. When the list is exhausted, the RUNOFF will stop.

The commands necessary to generate a form letter are:

```
(S)SELECT file-name (selection criteria) attribute-list

.READNEXT

.CHAIN item-name
```

The selected attribute-list contains all the variable information to be 'written' into the form letter. The use of '.READNEXT' commands reads each of these variables and causes them to be 'written' into the letter. The '.CHAIN' command causes the letter to be repeated so long as there is variable information in the selected attribute list. The following example demonstrates the generation of a form letter.

Assume the dictionary of the accounts payable file for a company contains the following three Attribute defining Items:

NAME	ACCOUNT	AMOUNT
001 A	001 A	001 A
002 1	002 2	003 3
003 CUSTOMER NAME	003 ACCOUNT TYPE	003 AMOUNT DUE
004	004	004
005	005	005
006	006	006
007	007	007
008 A1:","	008	008 A;3(MR2$,):"."
009 L	009 L	009 R
010 25	010 30	010 10

The dictionary also contains the following form letter written in RUNOFF:

```
    LETTER

   001 .SK 8
   002 Dear Mr.
   003 .READNEXT
   004  Our records show that your
   005 .READNEXT
   006 account is overdrawn by the amount of
   007 .READNEXT
   008 We would appreciate prompt payment.
   009  Thank you,
   010            Indiana Jones
   011 .SK 2
   012    President CELEBRITY SERVICES CO.
   013 .SK 3
   014 .BP
   015 .CHAIN LETTER
```

The data file contains items such as the following three:

```
    250                          251                   252

    001 Magic Johnson            001 Eddie Van Halen   001 Boy George
    002 Basketball Shoes         002 Guitar String     002 Voice Lesson
    003 25000                    003 12345             003 452359
```

To generate the form letter the data file is first sort selected by the name with the attribute list of NAME ACCOUNT and AMOUNT:

```
   SSELECT ACC-PAYABLE BY NAME WITH AMOUNT > "100" NAME ACCOUNT AMOUNT
```

This command will generated a selected list containing the following information:

```
    001 Boy George,
    002 Voice Lesson
    003 $4,523.59.
    004 Eddie Van Halen,
    005 Guitar String
    006 $123.45.
    007 Magic Johnson,
    008 Basketball Shoes
    009 $250.00.
```

Note that the correlatives on the names and on the amounts have been performed. Now by issuing the following RUNOFF command the form letters are generated:

```
    RUNOFF DICT ACC-PAYABLE LETTER (P)
```

The form letters will be printed as follows:

Dear Mr. Boy George,

Our records show that your Voice Lesson account is overdrawn by the amount of $4,523.59. We would appreciate prompt payment.

Thank You,

Indiana Jones

President CELEBRITY SERVICES CO.

(next page)

Dear Mr. Eddie Van Halen,

Our records show that your Guitar String account is overdrawn by the amount of $123.45. We would appreciate prompt payment.

Thank You,

Indiana Jones

President CELEBRITY SERVICES CO.

(next page)
Dear Mr. Magic Johnson,

Our records show that your Basketball Shoes account is overdrawn by the amount of $250.00. We would appreciate prompt payment.

Thank You,

Indiana Jones

President CELEBRITY SERVICES CO.

8.3.37 **SAVE INDEX file-name**

This command causes chapter and page number information of indexed data in a text to be saved in a separate file. Each indexed datum is stored as an individual item using the datum as the Item-Id, the chapter (where that datum is referenced) as the first attribute and the page number as the second attribute. Multiple values are stored in these attributes as multiple references to the same indexed datum are encountered. The resulting file may then be operated on by the ACCESS processor to generate listings for the chapter and page number information of all indexed data in a text.

The 'file-name' is the name of the file in which the chapter and page information is to be stored. NOTE: This must be a SEPARATE FILE from the text file !!! (Otherwise data in the text file will be DESTROYED.) The is placed in the text item itself and must precede the '.INDEX' commands. In short, only that indexed data which has been preceded by the '.SAVE INDEX' command will be saved in the specified file.

8.3.38 **SECTION n text**

This command may be used in conjunction with the CHAPTER command to handle automatic chapter section numbering and formatting. The

SECTION command automatically starts the next section at depth n, where n is the range 1-5. The text is printed following the section number SKIP occurs. The text is recorded as the section heading in the TABLE OF CONTENTS. If no text appears on the SECTION command, then no SKIP occurs and the section is not recorded in the TABLE OF CONTENTS. Section numbers are incremented automatically and the section number is printed in the form i.j.k.l.m with n digits printed.

Conventionally the .SECTION command is followed by a blank line before the next paragraph starts. Since the SECTION command causes a break which terminates the preceding paragraph, and since the text following the SECTION command is placed immediately into an ouput line and output prior to a consideration of the next line, the blank line after the SECTION command can be avoided by not indenting the first line of the next paragraph. That is, if the processor does not know that the next line starts a paragraph, it will not skip a line. It may be necessary to use an INDENT MARGIN if paragraph indentation is desired, however.

8.3.39 **SET TABS n,n,n, ...**

This command clears previous tab stops and sets new tab stops as indicated by the numeric tab positions. The tab stops (up to 30) must be greater than zero and in increasing order. They indicate tab stop positions relative to the left margin. Tabs are only in effect in NOFILL mode. The left-tab character (<) causes the next word to start at the next tab position. The right-tab character (>) causes the next word to end at the

next tab position. If a tab character appears at a point in the line where no further tab stops have been set, the tab character is ignored.

8.3.40 SKIP n (SK)

The SKIP causes a BREAK after which n*(SPACING n) lines are left blank. If the skip would advance past the end of the page, the output is advanced to the top of the next page. If n is missing, n=1 is assumed.

8.3.41 SPACE n (SP)

This command has the same affect as SKIP, except that n (rather than SPACING n) lines are left blank. SPACE is used where space is to be left independent of the line spacing; SKIP is used where space should be relative to the SPACING command. If n is missing, n=1 is assumed.

8.3.42 SPACING n

This command sets the line spacing to n. The command .SPACING 2 may be used for double spacing.

8.3.43 STANDARD

This command sets the standard (default) parameters and modes. STANDARD command is equivalent to the following commands:

```
.CS.F.J.UC.LEFT MARGIN 0.CRT.HEADING
.FOOTING
.PARAGRAPH 5.LINE LENGTH 74
```

8.3.44 TEST PAGE n

This command causes a BREAK followed by an advance to a new page when there are less than n lines remaining on the current page. If there are n or more lines remaining on the current page, this command has no effect. This command should be used to ensure that the following n lines are all output on the same page.

8.3.45 UPPER CASE (UC)

This command puts RUNOFF into upper case mode. Alphabetic letters will be processed as they are, unless modified by special commands or control characters. This command allows users of terminals with upper and lower case to generate the input text file without special commands or control characters. UC is one of the STANDARD settings. (See the STANDARD command.) The 'U' option will force the whole runoff output to upper-case if that is desired.

8.4 SPECIAL CONTROL CHARACTERS

RUNOFF features Special Control Characters for Upper/Lower Case Control, Underlining, Boldface Printing, Tabbing, and Special Character Override.

8.4.1 Upper and Lower Case Controls

The upper-case, lower-case control structure causes the text to go to the case specified. ENDCASE or EC will turn off both the upper-case condition and the lower-case condition to allow the text to go to its natural condition.

The forms ^^ and \\ cause the text to switch to upper-case or to lower-case in the same way that UC and LC cause the switch, except that ^^ and \\ may be imbedded in a line. Turning off the condition ^^ or \\ requires the use of EC.

The forms ^, \, & and @ will produce one character of upper-case, lower-case, underline, or overstrike. Each will be treated as the character itself if it is followed by a blank. The backarrow or underline character, _, will cause the succeeding character to be taken as a text character rather than a control character. This means that if you have existing RUNOFF text with forms such as '40# @$1.28/#', the dollar sign will be overstruck and the @ will disappear unless the backarrow, _, is inserted in front of the @. The same is true of the '&' character if it occurs in a character string.

The example below is an attempt to display the interactions of the several commands above. The first part is the text which was sent to RUNOFF and the second part is the output from RUNOFF. First, note that the 'I' in 'is' is always capitalized by the single-character ^, and that the 'a' is always in lower-case due to the single-character \ command.

The first line is in its natural form. The second line is uniformly capitalized by the UC command, excepting the 'a'. The third line is uniformly sent to lower-case, except for the 'Is'. The fourth and fifth lines contain a '^^ text \\ ' string, which is uniformly capitalized, excepting the 'a'. After the \\ the string reverts to lower-case. The only way to retrieve the capitalization of the string 'UC AND 'LC' is by the use of EC command. Thus, the sixth line is in its natural form.

```
001 .LINE LENGTH 66.PARAGRAPH 5.J
002 This ^is \a test of UC AND LC.
003 .UC
004 This ^is \a test of UC AND LC.
005 .LC
006 This ^is \a test of UC AND LC.
007 This ^is \a ^^test of UC AND LC.
008 This ^is \a test of UC AND LC.
009 .EC
010 This ^is \a test of UC AND LC.


     This Is a test of UC AND LC.

     THIS IS a TEST OF UC AND LC.

     This Is a test of uc and lc.

     This Is a TEST OF UC AND LC.

     THIS IS a TEST of uc and lc.

     This Is a test of UC AND LC.
```

Example of .UC, .LC, .EC and the associated ^ and \ characters.

8.4.2 Underlining and Boldface Printing

UNDERLINING

The ampersand (&) may be used to indicate underlining. The ampersand causes the letter immediately following to be underlined.

```
.LC
THE LETTER &A IS FIRST IN THE ALPHABET
```

This example of RUNOFF source would print as:

the letter a is first in the alphabet

Ampersand may be used in conjunction with the up-arrow and back-slash to underline a series of characters. An ampersand followed immediately by an up-arrow (&^) puts RUNOFF in the Underline mode until an ampersand followed immediately by a back-slash (&\) is encountered.

```
.UC
&^SPECIAL CONTROL CHARACTERS&\ ARE NEEDED ...
```

This example of RUNOFF source would print as:

SPECIAL CONTROL CHARACTERS ARE NEEDED ...

BOLDFACE PRINTING

The at sign (@) may be used to indicate **BOLDFACE** type. An at sign followed immediately by an up-arrow (@^) puts RUNOFF in the boldface mode until an at sign followed immediately by a back-slash (@\) is encountered. The number of times the boldface letters are overprinted may be set by using the numeric option of the RUNOFF verb.

```
.UC
@^SPECIAL CONTROL CHARACTERS@\ ARE NEEDED ...
```

This example of RUNOFF source would print as:

SPECIAL CONTROL CHARACTERS ARE NEEDED ...

The S option suppresses underlining and boldface printing.

8.4.3 Tab Settings

The less-than (<) and greater-than (>) characters may be used for tabbing. The left-tab character (<) causes the next word to start at the next tab position as set by the .SET TABS command. The right-tab character (>) causes the next word to end at the next tab position.

```
.NF
.SET TABS 5,8,25
.SK 1
><&^NAME<CONVENTIONAL DATA PROCESSING NAME&\
.SK 1
>1.<Item<Record
>1a.<Attribute<Field
>1b.<Item-id<Record Key
```

This example of RUNOFF source would print as:

	NAME	CONVENTIONAL DATA PROCESSING NAME
1.	Item	Record
1a.	Attribute	Field
1b.	Item-id	Record Key

Note: Tab characters are only in effect in the NOFILL (NF) mode.

You will also note that the sequence .tl .tl .tl will tab over to the third tab if tabs are set.

8.4.4 Special Character Override

The back-arrow or underscore (_) may be used to quote one of the special control characters or blanks. the letter immediately following the back-arrow is transmitted to the output without special processing.

```
_^SPECIAL _^CONTROL _^CHARACTERS ARE NEEDED ...
```

This example of RUNOFF source would print as:

```
^SPECIAL ^CONTROL ^CHARACTERS ARE NEEDED ...
```

✓ PPG 4
6/14/88

Chapter 9

PICK/BASIC

THE PICK SYSTEM

USER MANUAL

Contents

9 PICK/BASIC

9.1 THE PICK/BASIC LANGUAGE 9-5
9.2 PICK/BASIC LANGUAGE DEFINITIONS 9-7
9.3 PICK/BASIC FILE STRUCTURE 9-11
9.4 THE PICK/BASIC PROGRAM 9-12
9.4.1 DYNAMIC ARRAYS - FILE ITEM STRUCTURE 9-13
9.5 CREATING AND COMPILING PICK/BASIC PROGRAMS 9-15
9.5.1 PICK/BASIC COMPILER OPTIONS: A, C, E, L AND P OPTIONS 9-17
9.5.2 PICK/BASIC COMPILER OPTIONS : M, S, AND X OPTIONS 9-19
9.6 EXECUTING PICK/BASIC PROGRAMS 9-20
9.7 CATALOG AND DECATALOG : CREATING VERBS 9-21
9.7.1 PICK/BASIC EXECUTION FROM PROC 9-22
9.8 VARIABLES AND CONSTANTS : DATA REPRESENTATION 9-23
9.9 ARITHMETIC EXPRESSIONS 9-25
9.10 STRING EXPRESSIONS 9-27
9.11 RELATIONAL EXPRESSIONS 9-29
9.12 MATCHES : RELATIONAL EXPRESSION PATTERN MATCHING . . . 9-31
9.13 OR - AND : LOGICAL EXPRESSIONS 9-33
9.14 NUMERIC MASK AND FORMAT MASK CODES : VARIABLE FORMATTING 9-35
9.15 @ FUNCTION : CURSOR CONTROL 9-38
9.16 ABORT STATEMENT : TERMINATION 9-41
9.17 ABS FUNCTION : ABSOLUTE NUMERIC VALUE 9-42
9.18 ALPHA FUNCTION : ALPHABETIC STRING DETERMINATION . . . 9-43
9.19 ASCII FUNCTION : FORMAT CONVERSION 9-44
9.20 ASSIGNMENT STATEMENT : ASSIGNING VARIABLE VALUES . . . 9-45
9.21 BREAK ON AND OFF : DEBUGGER INHIBITION 9-46
9.22 CALL AND SUBROUTINE STATEMENTS : EXTERNAL SUBROUTINES . 9-47
9.22.1 ARRAY PASSING AND THE CALL STATEMENT 9-48
9.23 CASE STATEMENT : CONDITIONAL BRANCHING 9-49
9.24 CHAIN STATEMENT : INTERPROGRAM COMMUNICATION 9-50
9.25 CHAR FUNCTION : FORMAT CONVERSION 9-52
9.26 CLEAR STATEMENT : INITIALIZING VARIABLE VALUES 9-53
9.27 CLEARFILE STATEMENT : DELETING DATA 9-54
9.28 COL1() AND COL2() FUNCTIONS : STRING SEARCHING 9-55
9.29 COMMON STATEMENT : VARIABLE SPACE ALLOCATION 9-56
9.30 COS FUNCTION : COSINE OF AN ANGLE 9-58
9.31 COUNT FUNCTION : DYNAMIC ARRAYS 9-59
9.32 CRT STATEMENT : Terminal Output 9-60
9.33 DATA STATEMENT : STACKING INPUT DATA 9-61
9.34 DATE() FUNCTION : DATE CAPABILITY 9-63
9.35 DCOUNT FUNCTION : DYNAMIC ARRAYS 9-64
9.36 DELETE STATEMENT : DELETING ITEMS 9-65
9.37 DELETE FUNCTION : DYNAMIC ARRAY DELETION 9-66
9.38 DIM STATEMENT : DIMENSIONING ARRAYS 9-67
9.39 DTX FUNCTION : DECIMAL to HEXADECIMAL CONVERSION . . . 9-69
9.40 EBCDIC FUNCTION : FORMAT CONVERSION 9-70
9.41 ECHO ON AND OFF : TERMINAL DISPLAY 9-71
9.42 END STATEMENT 9-72
9.43 ENTER STATEMENT : INTERPROGRAM TRANSFERS 9-73
9.44 EQUATE STATEMENT : VARIABLE ASSIGNMENT 9-74
9.45 EXECUTE STATEMENT : EXECUTING TCL COMMANDS 9-75
9.45.1 INPUT - EXECUTE STATEMENT 9-75
9.45.2 OUTPUT - CAPTURING CLAUSE 9-75

9.45.3 OUTPUT - RETURNING CLAUSE 9-75
9.45.4 SELECT LISTS - EXECUTE STATEMENT 9-76
9.45.5 WORK ENVIRONMENT CHANGES 9-76
9.45.6 EXECUTE WORKSPACE 9-76
9.46 EXP FUNCTION : EXPONENTIAL CAPABILITY 9-77
9.47 EXTRACT FUNCTION : DYNAMIC ARRAY EXTRACTION 9-78
9.48 FIELD FUNCTION : STRING SEARCHING 9-80
9.49 FOOTING STATEMENT : PAGE OUTPUT FOOTINGS 9-81
9.50 FOR...NEXT STATEMENT : PROGRAM LOOPING 9-83
9.50.1 FOR...NEXT STATEMENT : EXTENDED PROGRAM LOOPING . . . 9-85
9.51 GOSUB AND ON...GOSUB STATEMENTS : SUBROUTINE BRANCHING 9-87
9.52 GOTO STATEMENT : UNCONDITIONAL BRANCHING 9-88
9.53 HEADING STATEMENT : PAGE OUTPUT HEADINGS 9-89
9.54 ICONV FUNCTION : INPUT CONVERSION 9-91
9.55 IF STATEMENT : SINGLE-LINE CONDITIONAL BRANCHING . . . 9-92
9.56 IF STATEMENT : MULTI-LINE CONDITIONAL BRANCHING 9-94
9.57 IN Statement - Single Character Input 9-96
9.58 INCLUDE STATEMENT : INCLUDING OTHER PICK/BASIC PROGRAMS 9-91
9.59 INDEX FUNCTION : SEARCHING FOR SUB-STRINGS 9-92
9.60 INPUT STATEMENT : TERMINAL INPUT 9-93
9.60.1 Using Masks with Input Statement 9-93
9.61 INPUTERR - INPUTTRAP - INPUTNULL : INPUT FORMS 9-95
9.62 INSERT FUNCTION : DYNAMIC ARRAY INSERTION 9-96
9.63 INT FUNCTION : INTEGER NUMERIC VALUE 9-97
9.64 LEN FUNCTION : GENERATING A LENGTH VALUE 9-98
9.65 LN FUNCTION : NATURAL LOGARITHM 9-99
9.66 LOCATE STATEMENTS : LOCATING INDEX VALUES 9-100
9.67 LOCK STATEMENT : SETTING EXECUTION LOCKS 9-102
9.68 LOOP STATEMENT : STRUCTURED LOOPING 9-104
9.69 MAT - ASSIGNMENT AND COPY : ASSIGNING ARRAY VALUES . . 9-106
9.70 MATREAD STATEMENT : MULTIPLE ATTRIBUTES 9-108
9.71 MATREADU STATEMENT : GROUP LOCKS 9-109
9.72 MATREADU STATEMENT : LOCKED CLAUSE 9-110
9.73 MATWRITE STATEMENT : MULTIPLE ATTRIBUTES 9-111
9.74 MATWRITEU STATEMENT : UPDATE LOCKS 9-112
9.75 NOT FUNCTION : LOGIC CAPABILITY 9-113
9.76 NULL STATEMENT : NON-OPERATION 9-114
9.77 NUM FUNCTION : NUMERIC STRING DETERMINATION 9-115
9.78 OCONV FUNCTION : OUTPUT CONVERSIONS 9-116
9.79 ON...GOTO STATEMENT : COMPUTED BRANCHING 9-117
9.80 OPEN STATEMENT : OPENING I/O FILES 9-118
9.81 OUT Statement - Single Character Output 9-119
9.82 PAGE STATEMENT : HEADING OUTPUT 9-119
9.83 PRECISION DECLARATION : SELECTING NUMERIC PRECISION . . 9-120
9.84 PRINT STATEMENT : TERMINAL OR PRINTER OUTPUT 9-122
9.85 PRINT STATEMENT : TABULATION AND CONCATENATION 9-124
9.86 PRINTER ON/OFF STATEMENTS : SELECTING OUTPUT DEVICE . . 9-125
9.87 PROCREAD STATEMENT : READING DATA FROM A CALLING PROC . 9-126
9.88 PROCWRITE STATEMENT : WRITING DATA BACK TO PROC 9-127
9.89 PROMPT STATEMENT : INPUT PROMPT CHARACTER 9-128
9.90 PWR FUNCTION : RAISING BY A POWER 9-129
9.91 READ STATEMENT : ACCESSING FILE ITEMS 9-130
9.92 READNEXT STATEMENT : ACCESSING ITEM-IDS 9-131
9.93 READT STATEMENT : READING RECORDS FROM TAPE 9-132
9.94 READU AND READVU STATEMENTS : GROUP LOCKS 9-133
9.95 READU AND READVU STATEMENTS : LOCKED CLAUSE 9-135

9.96 READV STATEMENT : ACCESSING AN ATTRIBUTE 9-136
9.97 RELEASE STATEMENT : RELEASING GROUP UPDATE LOCKS . . . 9-137
9.98 REM OR MOD FUNCTION : REMAINDER VALUE 9-138
9.99 REPLACE FUNCTION : DYNAMIC ARRAY REPLACEMENT 9-139
9.100 RETURN AND RETURN TO STATEMENTS : SUBROUTINE RETURNING 9-140
9.101 REWIND STATEMENT : REWINDING THE TAPE 9-141
9.102 RND FUNCTION : RANDOM NUMBER GENERATION 9-142
9.103 SELECT STATEMENTS : SELECTING ITEM-IDS 9-143
9.104 SEQ FUNCTION : FORMAT CONVERSION 9-145
9.105 SIN FUNCTION : SINE OF AN ANGLE 9-146
9.106 SLEEP OR RQM STATEMENT : TIME ALLOCATION 9-147
9.107 SPACE FUNCTION : STRING SPACING 9-148
9.108 SQRT FUNCTION : SQUARE ROOT CABABILITY 9-149
9.109 STOP STATEMENT : TERMINATION 9-150
9.110 STR FUNCTION : GENERATING STRING VALUES 9-151
9.111 SYSTEM FUNCTION : CALLING PRE-DEFINED SYSTEM VALUES . . 9-152
9.112 TAN FUNCTION : TANGENT OF AN ANGLE 9-155
9.113 TIME() AND TIMEDATE() FUNCTIONS : TIME AND DATE 9-156
9.114 TRIM FUNCTION : DELETING EXTRANEOUS SPACES 9-157
9.115 UNLOCK STATEMENT : CLEARING EXECUTION LOCKS 9-158
9.116 WEOF STATEMENT : POSITIONING TAPE 9-159
9.117 WRITE STATEMENT : MODIFYING ITEMS 9-160
9.118 WRITET STATEMENT : WRITING RECORDS TO TAPE 9-161
9.119 WRITEU AND WRITEVU STATEMENTS : UPDATE LOCKS 9-162
9.120 WRITEV STATEMENT : UPDATING AN ATTRIBUTE 9-163
9.121 XTD FUNCTION : HEXADECIMAL TO DECIMAL CONVERSION 9-164
9.122 PICK/BASIC SYMBOLIC DEBUGGER : AN OVERVIEW 9-165
9.122.1 USING THE PICK/BASIC DEBUGGER : AN EXAMPLE 9-167
9.122.2 THE TRACE TABLE 9-169
9.122.3 PICK/BASIC DEBUGGER: THE B, D, AND K COMMANDS 9-170
9.122.4 E, G, AND N COMMANDS : DEBUGGER EXECUTION 9-171
9.122.5 SLASH '/' COMMAND : DISPLAYING AND CHANGING VARIABLES 9-172
9.122.6 VARIOUS DEBUGGER COMMANDS : ADDITIONAL FEATURES . . . 9-173
9.122.7 GENERAL CODING TECHNIQUES : HELPFUL HINTS 9-174
9.122.8 PROGRAMMING EXAMPLES: PYTHAG 9-176
9.122.9 PROGRAMMING EXAMPLES: GUESS 9-177
9.122.10 PROGRAMMING EXAMPLES: INV-INQ 9-178
9.122.11 PROGRAMMING EXAMPLES: FORMAT 9-179
9.122.12 PROGRAMMING EXAMPLES: LOT-UPDATE 9-181
9.123 SUMMARY OF PICK/BASIC STATEMENTS 9-184
9.124 BASIC INTRINSIC FUNCTION SUMMARY 9-188
9.125 BASIC COMPILER ERROR MESSAGES 9-190
9.126 BASIC RUN-TIME ERROR MESSAGES 9-192
9.127 LIST OF ASCII CODES 9-194
9.128 SUMMARY OF THE PICK/BASIC DEBUGGER COMMANDS 9-197
9.129 DEBUGGER MESSAGES 9-199

Tables

9-1 Cursor Control Characters 9-39

9.1 THE PICK/BASIC LANGUAGE

This manual describes the PICK/BASIC source language, which is an extended version of Dartmouth BASIC.

BASIC (Beginners All-Purpose Symbolic Instruction Code) is a simple yet versatile programming language suitable for expressing a wide range of problems. Developed at Dartmouth College in 1963, BASIC is a language especially easy for the beginning programmer to master. Extended PICK/BASIC has the following extraordinary features:

- Optional statement labels (i.e., statement numbers)
- Statement labels of any length
- Multiple statements on one line
- Computed GOTO statements
- Complex IF statements
- Multi-line IF statements
- Priority CASE statement selection
- String handling with variable length strings up to 32,267 characters
- External subroutine calls
- Direct and indirect calls
- Magnetic tape input and output
- Fixed point arithmetic with up to 14 digit precision
- ACCESS data conversion capabilities
- PICK file access and update capabilities
- File level or group level lock capabilities
- Pattern matching
- Dynamic arrays
- TCL command execution

ABORT	FOOTING	MAT	PROMPT	STOP
BREAK ON/OFF	FOR...NEXT	MATCHES	READ	SUBROUTINE
CALL	GO	MATREAD	READNEXT	UNLOCK
CASE	GOSUB	MATREADU	READV	WEOF
CHAIN	GOTO	MATWRITE	READT	WRITE
CLEAR	GO TO	MATWRITEU	READV	WRITEU
CLEARFILE	HEADING	NEXT	READVU	WRITET
COMMON	IF	NULL	RELEASE	WRITEV
CRT	INPUT	ON GOSUB	REM * !	WRITEVU
DATA	INPUTERR	ON GOTO	RETURN	PROCREAD
DELETE	INPUTNULL	OPEN	RETURN TO	PROCWRITE
DIM	INPUTTRAP	PAGE	REWIND	EXECUTE
END	INPUT@	PRECISION	RQM	INCLUDE
ENTER	LOCK	PRINT	PRINTER ON/OFF	
EQUATE	LOOP	SELECT	SLEEP	

PICK/BASIC Statements

@	DATE()	INDEX	PWR	SYSTEM
ABS	DCOUNT	INSERT	REM	TAN
ALPHA	DELETE	INT	REPLACE	TIME()
ASCII	DTX	LEN	RND	TIMEDATE()
CHAR	EBCDIC	LN	SEQ	TRIM
COL1()	EXP	LOCATE	SIN	XTD
COL2()	EXTRACT	NOT	SPACE	
COS	FIELD	NUM	SQRT	
COUNT	ICONV	OCONV	STR	

PICK/BASIC Intrinsic Functions

9.2 PICK/BASIC LANGUAGE DEFINITIONS

A PICK/BASIC program is comprised of PICK/BASIC statements. PICK/BASIC statements may contain variables, constants, expressions, and PICK/BASIC Intrinsic Functions.

A PICK/BASIC program consists of a sequence of PICK/BASIC statements. More than one statement may appear on the same program line, separated by semicolons. Any PICK/BASIC statement may begin with an optional statement label.

PICK/BASIC statements may contain arithmetic, relational, and logical expressions. These expressions are formed by combining specific operators with variables, constants, or PICK/BASIC Intrinsic Functions. The value of a variable may change dynamically throughout the execution of the program. A constant, as its name implies, has the same value throughout the execution of the program. An Intrinsic Function performs a pre-defined operation upon the parameter(s) supplied.

FUNCTION	BRIEF DESCRIPTION
ABS	Returns an absolute value.
ALPHA	Tests for alphabetic value.
ASCII	Converts string from EBCDIC to ASCII.
CHAR	Converts numeric value to ASCII character.
COL1()	Returns column position preceding FIELD-selected sub-string.
COL2()	Returns column position following FIELD-selected sub-string.
COS	Generates the trigonometric cosine of an angle.
COUNT	Counts the number of occurrences in a string.
DATE()	Returns current internal date.
DCOUNT	Returns a value of the number of values in a string.
DELETE	Deletes attribute, value, or sub-value from dynamic array.
DTX	Converts decimal to hexadecimal.
EBCDIC	Coverts string from ASCII to EBCDIC.
EXP	The exponential function.
EXTRACT	Returns attribute, value, or sub-value from dynamic array.
FIELD	Returns a delimited sub-string.
ICONV	Provides for Pick input conversion.
INDEX	Returns column position of sub-string.
INSERT	Inserts attribute, value, or sub-value into dynamic array.
INT	Return an integer value.
LEN	Returns length of string.
LN	Generates the natural logarithm of the expression.
LOCATE	Returns the index of a sub-string in a dynamic array.
NOT	Returns logical inverse.
NUM	Tests for numeric value.
OCONV	Provides for Pick output conversion.
PWR	Raises an expression.

FUNCTION	BRIEF DESCRIPTION
REPLACE	Replaces attribute, value, or sub-value in dynamic array.
RND	Generates random number.
SEQ	Converts ASCII to a numeric value.
SIN	Generates trigonometric sine.
SPACE	Generates string containing blanks.
SQRT	Returns positive square root.
STR	Generates specified string.
SYSTEM	Provides certain pre-defined values.
TAN	Generates trigonometric tangent.
TIME	Returns internal time of day.
TIMEDATE	Returns external time and date.
TRIM	Removes extraneous blank spaces.
XTD	Converts hexadecimal to decimal.
@	Controls terminal cursor.

Summary of PICK/BASIC INTRINSIC FUNCTIONS

STATEMENT	BRIEF DESCRIPTION
BREAK ON/OFF	Enables or disables debugger.
CALL	External subroutine branching.
CASE	Provides conditional selection of a sequence of statements.
CHAIN	Passes control to another program.
CLEAR	Initializes all variables to zero.
CLEARFILE	Clears data section of specified file.
COMMON	Variable storage space allocation, used with CHAINed programs.
CRT	Directs output to the terminal.
DATA	Stores data for input using CHAIN or EXECUTE
DELETE	Deletes specified file item.
DIM	Reserves storage for arrays.
END	Designates the physical end of the program.
ENTER	Transfers control from one program to another
EQUATE	Allows variable to be defined as equivalent of another.
EXECUTE	Executes TCL commands.
FOR...NEXT	Specifies beginning of a program loop, NEXT specifies end.
GOSUB	Transfers control to a subroutine.
GOTO	Transfers control to another statement.
HEADING	Prints a page heading.
IF	Provides conditional execution of specified statements.
INCLUDE	Uses data from other programs.
INPUT	Inputs data from the terminal.
INPUTTER	Message is printed at bottom of screen.
INPUTNULL	Replaces default values with null.
INPUTTRAP	Sets input trap for character(s).
LOCK	Sets an execution lock.
LOOP...REPEAT	Provides for structured program loops.
MAT	Assigns value to each element of an array.
MATCHES	Relational pattern matching.
MATREAD	Reads a file item into an array.
MATREADU	Reads a file item into an array, sets update lock.
MATWRITE	Writes a file item with the contents of an array.
MATWRITEU	Same as MATWRITE but will not unlock update group.
NULL	Specifies a non-opertion.
ON GOTO/GOSUB	Transfers control using an indexed expression.
OPEN	Selects a file for subsequent I/O.
PAGE	Pages output device and prints heading.
PRECISION	Selects precision used in calculations.
PRINT	Causes specified data to be printed.
PRINTER ON/OFF	Controls selection of printer or terminal for output.
PROCREAD	Reads a PROC's primary input buffer.
PROCWRITE	Writes data to PROC's input buffer.
PROMPT	Selects a prompt character for the terminal.
READ	Reads a file item.
READU	Reads a file item, sets update lock.
READNEXT	Reads next item-id.

STATEMENT	BRIEF DESCRIPTION
READT	Reads next magnetic tape record.
READV	Reads an attribute.
READVU	Reads an attribute, sets update lock.
REM ! *	Specifies a remark (command) statement.
RETURN	Returns control from a subroutine.
RETURN TO	Return control to the main program
REWIND	Rewinds magnetic tape.
RQM or SLEEP	Terminates programs current time quantum.
SELECT	Selects data from a file.
STOP	Designates a logical end of the program.
SUBROUTINE	Specifies a program branch subroutine.
UNLOCK	Resets an execution lock.
WEOF	Writes an EOF on magnetic tape.
WRITE	Updates a file item.
WRITET	Writes a magnetic tape record.
WRITEU	Writes a file item, will not unlock update group.
WRITEV	Updates an attribute value.
WRITEVU	Updates an attribute value, will not unlock update group.

Summary of PICK/BASIC Statements

9.3 PICK/BASIC FILE STRUCTURE

A PICK/BASIC program, when stored, constitutes a File Item, and is referenced by its Item-Name (i.e., the name it is given when it is created via the EDITOR). An individual line within the PICK/BASIC program constitutes an attribute.

There is a fixed structure for PICK/BASIC source files. The file MUST have a dictionary and a separate data level. The PICK/BASIC source programs are stored in the data level of the file. The compiler writes the object and the symbol file as one record into the dictionary. This makes it much simpler to manipulate the program source. It can be LISTed, T-DUMPed, T-LOADed, and so on, without having to select the source items. The object record contains binary data, so the dictionary "D" pointer must have "DC" in attribute one. The primary advantages of this format are:

1. The object can now be protected with access/update locks.
2. The object saves/restores with the account on account-saves.
3. The CATALOG function is not necessary for run time efficiency.
4. There is less disk space utilized and fewer steps to perform.
5. The PICK/BASIC Debugger can tell the name of the item and verify the object code integrity.
6. PICK/BASIC has a restriction of 32267 bytes of object code and 32267 bytes of source per program.

A PICK/BASIC program is comprised of PICK/BASIC statements. The Remark statement may be used to identify the function or purpose of various sections of the program.

SEMICOLON - ';'

A PICK/BASIC program consists of a sequence of PICK/BASIC statements. More than one statement may appear on the same program line, separated by semicolons. For example:

```
X = 0;  Y = 0;  GOTO 50
```

LABELS - optional

Any PICK/BASIC statement may begin with an optional statement label which must be numeric only. A statement label is used so that the statement may be referenced from other parts of the program. For example:

```
100 INPUT X
169.40 INPUT Y
```

REMARKS - 'REM' '!' '*'

A helpful feature to use when writing a PICK/BASIC program is the Remark statement. A Remark statement is used to explain or document the program. It allows the programmer to place comments anywhere in the program without affecting program execution. A Remark statement is specified by typing the leters REM, or the asterisk character (*), or the exclamation (!) at the beginning of the statement; any arbitrary characters may then follow (up to the end of the line). For example:

```
REM THESE PICK/BASIC STATEMENTS
!   DO NOT AFFECT
*   PROGRAM EXECUTION
```

BLANK SPACES

Except for situations explicitly called out in the following sections, blank spaces appearing in the program line (which are not part of a data item) will be ignored. Thus, blanks may be used freely within the program for purposes of appearance.

```
      REM   PROGRAM TO PRINT NUMBERS
       !    FROM ONE TO MAX. NUMBER
            MAX NUM = 25; * define max number
       5*   FOR/NEXT LOOP ROUTINE
            FOR DSPLY = 1 to MAX. NUM
              PRINT DSPLY
            NEXT DSPLY
       9*   FINISHED
            END
```

Sample PICK/BASIC PRogram Including Remark Statements.

9.4.1 DYNAMIC ARRAYS - FILE ITEM STRUCTURE

PICK/BASIC allows the user to manipulate PICK file items in the form of item-formatted strings which are called dynamic arrays.

The PICK/BASIC language contains a number of statements and functions which are extremely useful in accessing and updating PICK files. A complete description of the PICK file structure is presented in the chapter on File-structure. A brief description of the structure as viewed by the PICK/BASIC programmer is appropriate at this point.

A PICK file consists of a set of items. When a PICK file item is accessed by a PICK/BASIC program (refer to INPUT/OUTPUT STATEMENTS), it is represented as a PICK/BASIC string in item format. A string in item format is called a dynamic array.

A dynamic array consists of one or more attributes separated by attribute marks (i.e., an attribute mark has an ASCII equivalent of 254, which prints as "^"). An attribute, in turn, may consist of a number of values separated by value marks (i.e, a value mark has an ASCII equivalent of 253, which prints as "]"). Finally, a value may consist of a number of secondary values separated by secondary value marks (i.e., a secondary value mark has an ASCII equivalent of 252, which prints as "\"). An example of a dynamic array is as follows:

"55^ABCD^73XYZ^100000.33"

where "55", "ABCD", "73XYZ", and "100000.33" are attributes.
The following illustrates a more complex dynamic array:

"Q5^AAAA^952]ABC]12345^A^B^C]TEST\12I\9\99.3]2^555"

where "Q5", "AAAA", "952]ABC]12345", "A", "B", "C]TEST]\12I\9\99.3]2" and "555" are attributes; "952", "ABC", "12345", "C", "TEST\12I\9\99.3", and "2" are values; and "TEST", "12I", "9", and "99.3" are secondary values.

Dynamic arrays can be directly manipulated by the PICK/BASIC dynamic array functions (refer to the section titled PICK/BASIC INTRINSIC FUNCTIONS). Dynamic arrays are called "arrays" because they can be referenced by these functions using 3 subscripts. They are "dynamic" in the sense that elements can be added and deleted without having to re-compile the program, as long as the item does not exceed 32,267 characters.

ARRAY	EXPLANATION
123^456^789]ABC]DEF	"123", "456", "789]ABC]DEF" are attributes; "789", "ABC" and "DEF" are values.
Q56^3.22]3.56\88\B2C^99	"Q56", "3.22]3.56\88\B2C", and "99" are attributes; "3.56/88/B2C" is a value; "3.56", "88", and "B2C" are secondary values.
A]B]C]D^E]F]G]H^I]J	"A]B]C]D", "E]F]G]H", and "I]J" are attributes; "A", "B", "C", "D", "E", "F", "G", "H", "I", and "J" are values.

Sample Usage of Dynamic Arrays.

9.5 CREATING AND COMPILING PICK/BASIC PROGRAMS

A PICK/BASIC program is created via the EDITOR as any other data-file item. Once this source code item has been filed, it is compiled by issuing the COMPILE verb (or the BASIC verb) at the TCL level.

FORMAT:

```
ED (or EDIT) file-name item-id
```

Upon execution, the EDITOR processor will then be entered, and the user may begin entering his PICK/BASIC program. For ease of instruction indentation, the user may set tab stops (either at the TCL level or while the EDITOR processor is in control-- see examples below).

The program name will be that specified by the 'item-id' and the program will be stored in the file specified by the 'file-name'. Users will typically have a file exclusively devoted to the storage of PICK/BASIC programs. The PICK/BASIC compiler stores the object code in the same file, but in the dictionary portion of the file (see below).

Once the PICK/BASIC program has been entered and filed, it may be compiled by issuing the BASIC verb at the TCL level. BASIC is a TCL-II verb which creates a new dict item: it contains the compiled PICK/BASIC program (the object code), and a symbol definition table of the variables used in the program. The item is stored in the file specified by 'file-name'.

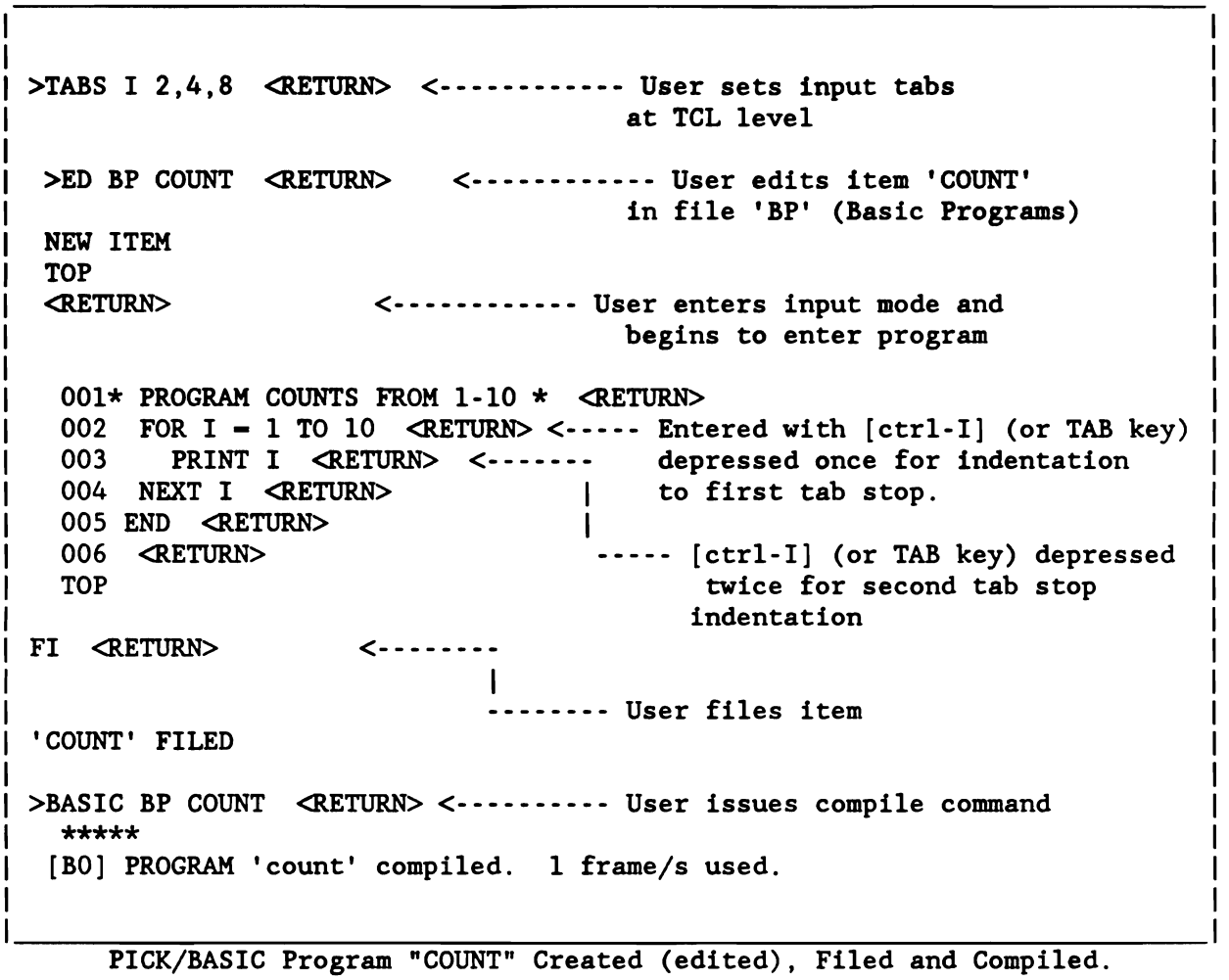
FORMAT:

```
BASIC file-name item-list {(options}
```

The 'item-list' consists of one or more item-id's (program names) separated by one or more blanks. The 'options' parameter is optional and if used, must be preceded by a left parenthesis. Multiple options should be separated by commas. Valid options are listed below. For detailed descriptions of each, see the following section.

BASIC VERB OPTIONS

A	**Assembled code option**
C	**Suppress End Of Line (EOL) opcodes from object code.**
E	**List error lines only.**
L	**List PICK/BASIC program.**
M	**List map of PICK/BASIC program**
P	**Print compilation output on line printer.**
S	**Suppress generation of symbol table.**
X	**Cross reference all variables.**



PICK/BASIC Program "COUNT" Created (edited), Filed and Compiled.

9.5.1 PICK/BASIC COMPILER OPTIONS: A, C, E, L AND P OPTIONS

This section describes five of the options available when issuing the BASIC compile statement. They are the "A" for assembled code, the "C" for suppression of end of line opcode, "E" for the listing of error lines only, the "L" for the listing of the program during compilation, and "P" for routing output to the printer. The following section describes the remaining three compiler options.

FORMAT:

```
BASIC file-name item-name ((options)
```

If multiple options are present, they are seperated by commas.

A - The assembled code option. The "A" option generates a listing of the source code line numbers, the labels and the PICK/BASIC opcodes used by the program. This is a 'pseudo' assembly code listing which allows the user to see what PICK/BASIC opcodes his program has generated. The hexadecimal numbers on the left of the listing are the PICK/BASIC opcodes and the mnemonics are listed on the right. The assembled code listing of the PICK/BASIC program "COUNT" (from previous section) is shown, as an example, on the facing page.

C - The compress option. The compress option suppresses the end-of-line (EOL) opcodes from the object code item. The EOL opcodes are used to count lines for error messages. This eliminates 1 byte from the run time object code for every line in the source code. This option is designed to be used with debugged programs. Any run time error message will specify a line number of 1.

E - The 'list error lines only' option. The "E" option generates a listing of the error lines encountered during the compilation of the program. The listing indicates line number in the source code item, the source line itself and a description of the error associated with the line.

L - The list program option. The "L" option generates a line by line listing of the program during compilation. Error lines with associated error messages are indicated.

P - The printer option. The "P" option routes all output generated by the compilation to the printer.

```
SOURCE     BASIC      PSEUDO
CODE       OBJECT     ASSEMBLY
LINE NO.   CODE       CODE

001        01         EOL
002        03         LOADA      I
002        07         LOAD 1
002        07         LOAD 1
002        2D         SUBTRACT
002        5F         STORE
002         *1009
002        05         LOADN      10
002        03         LOADA      I
002        07         LOAD 1
002        1B         FORNEXT    *2009
002        01         EOL
003        16         LOAD       I
003        50         PRINTCRLF
003        01         EOL
004        06         BRANCH     *1009
004         *2009
004        01         EOL
005        01         EOL
006        45         EXIT

[B0] Program 'count' compiled.  1 frame/s used.
```

"A" option listing of PICK/BASIC program "COUNT"

9.5.2 PICK/BASIC COMPILER OPTIONS : M, S, AND X OPTIONS

This section describes the remaining three options available when issuing the BASIC compile statement. They are "M" for map, "S" for suppressing generation of the symbol table, and "X" for cross reference.

M - The map option. The "M" option generates a variable map which is printed out after compilation. These maps show where the program data has been stored in the user's workspace. The variable map lists the offset in decimal of every PICK/BASIC variable in the program. For example, the form:

```
20 xxx 30 yyy
```

shows that the descriptor of variable 'xxx' starts on byte 20 and the descriptor of variable 'yyy' starts on byte 30 of the seventh frame of the I$ buffer. Descriptors are 10 bytes in length.

S - The suppress symbol table option. The "S" option suppresses the the symbol table item which is normally generated during compilation. The symbol table item is used exclusively by the PICK/BASIC DEBUGGER for reference, therefore it must be kept only if the user wishes to use the Debugger.

X - The cross reference option. The "X" option creates a cross reference of all the labels and variables used in a PICK/BASIC program and stores this information in the BSYM file. Note: A BSYM file must exist (a modulo and separation of 1,1 should be sufficient). The "X" option first clears the BSYM file information in the BSYM file then creates an item for every variable and label used in the program. The item-id is the variable or label name. The attributes contain the line numbers of where the variable or label is referenced. An asterisk will precede the line number where a label is defined, or where the value of the variable is changed.

No output is generated by this option. an attribute definition item should be placed in the dictionary of the "BSYM" file which allows a cross reference listing of the program to be generated by the command: >SORT BSYM BY LINE-NUMBER LINE-NUMBER

9.6 EXECUTING PICK/BASIC PROGRAMS

The PICK/BASIC program is executed by issuing the RUN verb.

FORMAT:

RUN file-name item-id {(options)}

RUN is the TCL-II verb issued to run a compiled PICK/BASIC program. The "file-name" and "item-id" specify the compiled PICK/BASIC program to be executed. The "options" parameter is optional (if used, it must be enclosed in parentheses). Multiple options are separated by commas. Valid options are as follows:

A - Abort option. The "A" option inhibits entry to the Basic Debugger under all error conditions; instead, the program will print a message and terminate execution.

D - Run-time debug option; causes the PICK/BASIC debugger to be entered before the start of program execution. Note that the PICK/BASIC debugger may also be called at any time while the program is executing, by pressing the BREAK key on the terminal.

E - Errors option. The "E" option forces the PICK/BASIC runtime package to enter the PICK/BASIC Debugger whenever an error condition occurs. The use of this option will force the operator to either accept the error by using the Debugger, or exit to TCL.

N - Nopage option. The "N" option cancels the default wait at the end of each page of output.

P - Printer on (has same effect as issuing a PICK/BASIC PRINTER ON statement). Directs all program output from a PRINT statement to the printer.

S - Suppress run-time warning messages.

```
                    TESTING

001 * PROGRAM TO PRINT TEST MESSAGE
002 PRINT "THIS IS A TEST"
003 END


  > RUN PROGRAMS TESTING <RETURN>
    THIS IS A TEST
```

Execution of Sample PICK/BASIC Program

9.7 CATALOG AND DECATALOG : CREATING VERBS FOR BASIC PROGRAMS

Verbs defining PICK/BASIC programs can be created and deleted using the CATALOG and DECATALOG verbs.

The CATALOG verb creates a TCL-II verb defining a PICK/BASIC program.

FORMAT:

```
CATALOG file-name item-id
```

The "file-name" and "item-id" specify the previously compiled PICK/BASIC program which is to be cataloged. The system will respond with:

```
[244] item-id CATALOGED
```

Once a program is cataloged, it is 'run' simply by issuing the program name at the TCL prompt. The TCL-II verb which is added to the user's Master Dictionary has the following form:

```
1)  P
2)  E6
3)
4)
5)  XXXXX
```

where XXXXX is the user's basic program file name.

The DECATALOG verb deletes the verb definition from the user's Master Dictionary.

FORMAT:

```
DECATALOG file-name item-id
```

9.7.1 PICK/BASIC EXECUTION FROM PROC

PICK/BASIC program execution can be initiated from PROC, similar to any other TCL command.

A PICK/BASIC program may be run from a PROC. The following example illustrates the use of a PICK/BASIC program in conjunction with the ACCESS Sort Select (SSELECT) verb.

PROC named LISTBT as follows:

```
PQ
HSSELECT BASIC/TEST
STON
HRUN BASIC/TEST LISTIDS
P
```

PICK/BASIC program named LISTIDS as follows:

```
10  N = 0
20  READNEXT ID ELSE STOP
    PRINT ID 'L#18':
    N = N + 1
    IF N>= 4 THEN PRINT; GO TO 10
    GO TO 20
    END
```

By typing in the following:

```
LISTBT
```

at the TCL level, the PROC LISTBT selects the item-id's contained in file BASIC/TEST and invokes the BASIC program LISTIDS to list the item-id's selected, four to a line.

Sample Usage of PICK/BASIC called from PROC.

9.8 VARIABLES AND CONSTANTS : DATA REPRESENTATION

There are two types of data: **NUMERIC** and **STRING**. These data types are represented within the PICK/BASIC program as either **variables** or **constants**.

Numeric data consists of a series of digits and represent an amount (e.g., 255). String data consist of a set of characters, such as would be for a name and address. For example:

```
JOE DOE, 430 MAIN, ATOWN, CA.
```

These data types may be represented within the PICK/BASIC program as either constants or variables. A constant, as its name implies, has the same value throughout the execution of the program. A numeric constant may contain up to 14 digits, including a maximum of 6 digits following the decimal point and must be in the range:

```
-99,999,999.999999 to 99,999,999.999999
```

If the PRECISION (see section on PRECISION DECLARATION) of the program is 6 digits; by setting the PRECISION to a value less than 6, the range of the allowable numbers is increased accordingly.

The unary minus sign is used to specify negative constants. For example:

```
-17000000
-14.3375
```

A string constant is represented by a set of characters enclosed in single quotes, double quotes, or backslashes. For example:

```
"THIS IS A STRING" 'ABCD1234#*' \HELLO\
```

if any of the string delimiters (',", or \) are to be part of the string, then one of the other delimiters must be used to delimit the string. For example:

```
"THIS IS A 'STRING' EXAMPLE"
\THIS IS A "STRING" EXAMPLE\
```

A string may contain from 0 to 32,267 characters.

As mentioned above, data may also be represented as variables. A variable has a name and a value. The value of a variable may be either numeric or string, and may change dynamically throughout the execution of the program. The name of a variable identifies the variable (the name remains constant throughout program execution). Variable names consist of an alphabetic character followed by zero or more letters, numerals, periods, or dollar signs. The length of a variable name may be from 1 to 64 characters.

For example:

```
X
QUANTITY
DATA.LENGTH
B$..$
```

The variable X, for example, may be assigned the value 100 at the start of a program, and may then later be assigned the value "THIS IS A STRING".

It should be noted that PICK/BASIC keywords (i.e., words that define PICK/BASIC statements and functions) may not be used as variable names.

VALID STRING	INVALID STRING
"ABC%123#*4AB"	ABC123 (i.e., quotes are missing)
'1Q2Z....'	'ABC%QQR" (either two single quotes or two double quotes must be used)
"A 'LITERAL' STRING"	
'A "LITERAL" STRING'	
'' (i.e., the empty string)	"12345678910 (terminating double quote missing)
\JOHN PROGRAMMER\	

Sample Usage of String Constants

VALID VARIABLE NAME	INVALID VARIABLE NAME
A5	ABC 123 (no space allowed)
ABCDEFGHI	5AB (must begin with letter)
QUANTITY.ON.HAND	Z.,$ (comma not allowed)
R$$$$P$	A-B ("-" not allowed)
J1B2Z	INPUT (Pick/Basic Statement)
INTEGER	
THIS.IS.A.NAME	

Sample Usage of Variable Names

9.9 ARITHMETIC EXPRESSIONS

Expressions are formed by combining operators with variables, constants, or PICK/BASIC Intrinsic Functions. Arithmetic expressions are formed by using arithmetic operators.

When an expression is encountered as part of a PICK/BASIC program statement, it is evaluated by performing the operations specified by each of the operators on the adjacent operands, i.e., the adjacent constants, identifiers, or Intrinsic Functions. (NOTE: Intrinsic Functions are discussed in a separate section of this manual.)

Arithmetic expressions are formed by using the arithmetic operators listed below. The simplest arithmetic expression is a single numeric constant, variable, or Intrinsic Function. A simple arithmetic expression may combine two operands using an arithmetic operator. More complicated arithmetic expressions are formed by combining simple expressions using arithmetic operators.

When more than one operator appears in an expression, certain rules are followed to determine which operation is to be performed first. Each operator has a precedence rating. In any given expression the highest precedence operation will be performed first. Precedence of the arithmetic operators are shown below. If there are two or more operators with the same precedence (or an operator appears more than once) the leftmost operation is performed first. For example, consider this expression: -R/A+B*C. The unary minus is evaluated first (i.e., -R = Result1). The expression then becomes: Result 1 / A+B*C. The division and multiplication operators have the same precedence; since the division operator is leftmost it is evaluated next (i.e., Result1 / A = Result2). The expression then becomes: Result 2+B*C. The multiplication operation is performed next (i.e., B*C = Result3). The Result2 + Result3 = Final Result.

Using some figures in the above expression illustrates, for example, that the expression -50/5+3*2 evaluates to -4.

Any sub-expression may be enclosed in parentheses. Within the parentheses, the rules of precedence apply. However, the parenthesized subexpression as a whole has highest precedence and is evaluated first. For example: (10+2)*(3-1) = 12*2 = 24. Parentheses may be used anywhere to clarify the order of evaluation, even if they do not change the order.

If a string value containing only numeric characters is used in an arithmetic expression, it is considered as a decimal number. For example, 123 + "456" evaluates to 579.

If a string value containing non-numeric characters is used in an arithmetic expression, a warning message will be printed (refer to APPENDIX D - PICK/BASIC RUN-TIME ERROR MESSAGES) and zero will be assumed for the string value.

The following expression, for example, evaluates to 123:

```
123 + "ABC"
```

OPERATOR SYMBOL	OPERATION	PRECEDENCE
+	unary plus	1 (high)
-	unary minus	1
^	exponental	2
*	multiplication	3
/	division	3
+	addition	4
-	subtraction	4 (low)

Arithmetic Operators

2+6+8/2+6	Evaluates to 18
12/2*3	Evaluates to 18
12/(2*3)	Evaluates to 2
2^2*3	Evaluates to 12
2^(2*3)	Evaluates to 64
A+75/25	Evaluates to 3 plus the current value of variable A.
-5+2	Evaluates to -3
-(5+2)	Evaluates to -7
8*(-2)	Evaluates to -16
5 * "3"	Evaluates to 15

Sample Usage of Arithmetic Expressions.

9.10 STRING EXPRESSIONS

A string expression may be: a string constant, a variable with a string value, a sub-string, or a concatenation of string expressions. String expressions may be combined with arithmetic expressions.

FORMAT:

variable[expression1,expression2]

expression1 - - - - starting character position

expression2 - - - - - number of characters in sub-string length

A sub-string is a set of characters which makes up part of a whole string. For example, "SO.", "123", and "ST." are sub-strings of the string "1234 SO. MAIN ST." Sub-strings are specified by a starting character position and a sub-string length, separated by a comma and enclosed in square brackets. For example, if the current value of variable S is the string "ABCDEFG", then the current value of S[3,2] is the sub-string "CD" (i.e., the two character sub-string starting at character position 3 of string S). Furthermore, the value of S[1,1] would be "A", and the value of S[2,6] would be "BCDEFG".

If the "starting character" specification is past the end of the string value, then an empty sub-string value is selected (e.g., if A has a value of 'XYZ', then A[4,1] will have a value of ''). If the "starting character" specification is negative or zero, then the first character is assumed (e.g., if X has a value of 'JOHN', the X[-5,1] will have a value of 'J').

If the "sub-string length" specification exceeds the remaining number of characters in the string, then the remaining string is selected (e.g., if B has a value of '123ABC', the B[5,10] will have a value of 'BC'). If the "sub-string length" specification is negative or zero, then an empty sub-string is selected (e.g., B[5,-2] and B[5,0] both have a value of '').

Concatenation operations may be performed on strings. Concatenation is specified by a colon (:) or CAT operator. The concatenation of two strings (or sub-strings) is the addition of the characters of the second operand onto the end of the first. For example:

```
"AN EXAMPLE OF " CAT "CONCATENATION"

                                    evaluates to:

"AN EXAMPLE OF CONCATENATION"
```

The precedence of the concatenation operator is higher than any of the arithmetic operators. So if the concatenation operator appears in the same expression with an arithmetic operator, the concatenation operation will be performed first. Multiple concatenation operations are performed from left to right. Parenthesized sub-expressions are evaluated first. The concatenation operator considers both its operands to be string values; for example, the expression 56:"ABC" evaluates to "56ABC":

NOTE: For the following examples, assume that the current value of A is "ABC123", and the current value of variable Z is "EXAMPLE".

EXPRESSION	EXPLANATION
Z[1,4]	Evaluates to "EXAM".
A:Z[1,1]	Evaluates to "ABC123E".
Z[1,1] CAT A[4,3]	Evaluates to "E123"
5*2:0	2:0 is evaluated first and results in the string "20" (i.e., concatenation operator assumes both operands are strings). 5*"20" is then evaluated and results in 100 (i.e., * operator assumes both operands are numeric. Final result is 100.
A[6,1]+5	Evaluates to 8.
Z CAT A:Z	Evaluates to "EXAMPLEABC123EXAMPLE".
Z CAT " ONE"	Evaluates to "EXAMPLE ONE".

Examples of String Expressions Combined With Arithmetic Examples.

9.11 RELATIONAL EXPRESSIONS

Relational expressions are the result of applying a relational operator to a pair of arithmetic or string expressions.

The relational operators are listed below. A relational operation evaluates to 1 if the relation is true, and evaluates to 0 if the relation is false. Relational operators have lower precedence than all arithmetic and string operators; therefore, relational operators are only evaluated after all arithmetic and string operations have been evaluated.

For purposes of clarification, relational expressions may be divided into two types: arithmetic relations and string relations. An arithmetic relation is a pair of arithmetic expressions separated by any one of relational operators. For example:

3 < 4	(3 is less than 4)=(true)=1
3 = 4	(3 is equal to 4)=(false)=0
3 GT 3	(3 is greater than 3)=(false)=0
3 >= 3	(3 is greater than or equal to 3)=(true)=1
5+1 > 4/2	(5 plus 1 is greater than 4 divided by 2)=(true)=1

A string relation is a pair of string expressions separated by any one of the relational operators. A string relation may also be a string expression and an arithmetic expression separated by a relational operator (i.e., if a relational operator encounters one numeric operand and one string operand, it treats both operands as strings). To resolve a string relation, character pairs (one from each string) are compared one at a time from leftmost characters to rightmost. If no unequal character pairs are found, the strings are considered to be 'equal'. If an unequal pair of characters are found, the characters are ranked according to their numeric ASCII code equivalents (refer to the LIST OF ASCII CODES in APPENDIX E of this manual). The string contributing the higher numeric ASCII code equivalent is considered to be "greater" than the other string. Consider the following relation:

```
"AAB" > "AAA"
```

This relation evaluates to 1 (true) since the ASCII equivalent of B (66) is greater than the ASCII equivalent of A (65).

If the two strings are not the same length, but the shorter string is otherwise identical to the beginning of the longer string, then the longer string is considered "greater" than the shorter string. The following relation, for example, is true and evaluates to 1:

```
"STRINGS" GT "STRING"
```

OPERATOR SYMBOLS	OPERATION
< or LT	Less than
> or GT	Greater than
<= or LE or =<	Less than or equal to
>= or GE or =>	Greater than or equal to
= or EQ	equal to
# or < > or < > or NE	not equal to
MATCH or MATCHES	pattern matching (see next page)

Relational Operators

EXPRESSION	EXPLANATION
4 < 5	Evaluates to 1 (true).
"D" EQ "A"	Evaluates to 0 (false).
"D" > "A"	ASCII equivalent of D (X'44') is greater than ASCII equivalent of A (X'41'), so expression evaluates to 1.
"Q" LT 5	ASCII equivalent of Q (X'51') is not less than ASCII equivalent of 5 (X'35'), so expression evaluates to 0.
6+5 = 11	Evaluates to 1.
Q EQ 5	Evaluates to 1, if current value of variable Q is 5; evaluates to 0 otherwise.
"ABC" GE "ABB"	Evaluates to 1 (i.e., C is "greater" than B).
"XXX" LE "XX"	Evaluates to 0.

Sample Usage of Relational Expressions.

9.12 MATCHES : RELATIONAL EXPRESSION PATTERN MATCHING

BASIC pattern matching allows the comparison of a string value to a predefined pattern. Pattern matching is specified by the MATCH or MATCHES relational operator.

FORMAT:

```
expression MATCH(ES) "pattern"
```

The MATCH or MATCHES relational operator compares the string value of the expression to the predefined pattern (which is also a string value) and evaluates to 1 (true) or 0 (false). The pattern may consist of any combination of the following:

- An integer number followed by the letter N (which tests for that number of numeric characters).
- An integer number followed by the letter A (which tests for that number of alphabetic characters).
- An integer number followed by the letter X (which tests for that number of any characters).
- A literal string enclosed in quotes (which tests for that literal string of characters).

Consider the following expression:

```
DATA MATCHES "4N"
```

This relation evaluates to 1 if the current string value of variable DATA consists of four numeric characters.

If the integer number used in the pattern is 0, then the relation will evaluate to true only if all the characters in the string conform with the "specification letter" (i.e., N,A, or X). For example:

```
X MATCH "0A"
```

This relation evaluates to 1 if the current string value of variable X consists only of alphabetic characters.

As a further example, consider the following expression:

```
A MATCHES "1A4N"
```

This relation evaluates to 1 if the current string value of variable A consists of an alphabetic character followed by four numeric characters.

EXPRESSION	EXPLANATION
Z MATCHES '9N'	Evaluates to 1 if current string value of variable Z consists of 9 numeric characters; evaluates to 0 otherwise.
Q MATCHES "ON"	Evaluation to 1 if current value of Q is any unsigned integer; evaluates to 0 otherwise.
B MATCH '3N"-"2N"-"4N'	Evaluates to 1 if current value of B is, for example, any social security number; evaluates to 0 otherwise.
B="4N1A2N" C MATCHES B	Evaluates to 1 if current string value of C consists of four numeric characters followed by one alphabetic character followed by two numeric characters; evaluates to 0 otherwise.
A MATCHES "ON'.'ON"	Evaluates to 1 if current value of A is any number containing a decimal point; evaluates to 0 otherwise.
"ABC" MATCHES "3N"	Evaluates to 0.
"XYZ" MATCHES "0A"	Evaluates to 1.
"XYZ1" MATCH "4X"	Evaluates to 1.
X MATCHES ''	Evaluates to 1 if current string value of X is the empty string; evaluates to 0 otherwise.

Sample Usage of Pattern Matching Relation.

9.13 OR - AND : LOGICAL EXPRESSIONS

Logical expressions (also called Boolean expressions) are the Result of applying logical (Boolean) operators to relational or arithmetic expressions.

FORMAT:

AND	or	&	Logical And operation
OR	or	!	Logical Or operation

Logical operators operate on the true or false Results of relational or arithmetic expressions. (Relational expressions are considered false when equal to zero, and are considered true when equal to one; arithmetic expressions are considered false when equal to zero, and are considered true when not equal to zero.) Logical operators have the lowest precedence and are only evaluated after all other operations have been evaluated. If two or more logical operators appear in an expression, the leftmost is performed first.

Logical operators act on their associated operands as follows:

a OR b is true (evaluates to 1) if a is true or b is true; is false (evaluates to 0) only when a and b are both false.

a AND b is true (evaluates to 1) only if both a and b are true; is false (evaluates to 0) if a is false or b is false or both are false.

consider, for example, the following logical expression:

A*2-5>B AND 7>J

The multiplication operation has highest precedence, so it is evaluated first (i.e., A*2 = Result1). the expression then becomes:

Result1 - 5>B AND 7>J

The subtraction operation is next (i.e., Result1 - 5=Result2). The expression then becomes:

Result2 > B AND 7>J

the two relational operators are of equal precedence, so the leftmost is evaluated first (i.e., Result2 > B=Result3, where Result3 has a value of 1 indicating true, or a value of 0 indicating false). the expression then becomes:

Result3 AND 7>J The remaining relational operation is then performed (i.e., 7>J = Result 4, where Result4 equals 1 or 0). The final expression therefore becomes:

Result3 AND Result4

which is evaluated as true (1) if both Result3 and Result4 are true, and is evaluated as false (0) otherwise.

The NOT function may be used in logical expressions to negate (invert) the expression or sub-expression (refer to the description of the NOT Intrinsic Function).

EXPRESSION	EXPLANATION
1 AND A	Evaluates to 1 if current value of variable A is non-zero; evaluates to 0 if current value of A is 0.
8-2*4 OR Q5-3	Evaluates to 1 if current value of Q5-3 is non-zero; evaluates to 0 if current value of Q5-3 is 0.
A>5 OR A<0	Evaluates to 1 if the current value of variable A is greater than 5 or is negative; evaluates to 0 otherwise.
1 AND (0 OR 1)	Evaluates to 1.
J EQ 7 AND I EQ 5*2	Evaluates to 1 if the current value of variable J is 7 and the current value of variable I is 10; evaluates to 0 otherwise.
"XYZ1" MATCH "4X" AND X	Evaluates to 1 if the current value of variable X is non-zero; evaluates to 0 if current value of X is 0.
X1 AND X2 AND X3	Evaluates to 1 if the current value of each variable (X1, X2, and X3) is non-zero; evaluates to 0 if the current value of any or all variables is 0.

Sample Usage of Logical Expressions.

9.14 NUMERIC MASK AND FORMAT MASK CODES : VARIABLE FORMATTING

Variable values may be formatted via the use of format strings. A format string immediately following a variable name or expression specifies that the value will be formatted as specified by the characters within the format string. The format string may also be used directly in conjunction with the PRINT statement.

FORMAT:

variable = variable"{numeric mask code}{(format mask code)}"

The format string uses the same subroutines as the ACCESS Mask Conversion Code. It may be used to format both numeric and non-numeric strings.

The entire format string is enclosed in quotes. If the format mask is used, it is enclosed in parentheses within the quotes.

The entire format string may be used as a literal, or it may be assigned to a variable. In either case the format string or variable immediately follows the variable it is to format.

The numeric mask code is represented by the symbols: j, n, m, Z, ',', c and $, which controls justification, precision, scaling and credit indication. The format mask code controls field length and fill characters.

The formatted value may be assigned to the same variable or to a new variable (as shown in the general form), or it may be used in a PRINT statement of the form: PRINT X"format string".

The format mask code may be used separately or in conjunction with the numeric mask.

The format mask code is enclosed in parentheses, and may consist any combination of format characters and literal data.

The field length specified ('n') should not exceed 99. The format characters are "#", "*" or "%", optionally followed by a numeric, such as "#3" or "%5".

Any other character in the format field, including parentheses, may be used as a literal character.

NOTE: If a dollar sign is placed outside of the format mask, it will be output just prior to the value, regardless of the filled mask. If a dollar sign is used within the format field it will be output in the leftmost position regardless of the filled field.

NUMERIC MASK CODE:

j specifies justification. "R" specifies right justification. "L" specifies left justification. Default justification is left.

n is a single numeric digit defining the number of digits to print out following the decimal point. If n = 0, the decimal point will not be output following the value.

m is an optional 'scaling factor' specified by a single numeric digit which 'descales' the converted number by the 'mth' power of 10. Because PICK/BASIC assumes 4 decimal places (unless otherwise specified by a Precision Statement) to descale a number by 10 m should be set to 5, to descale a number by 100, m should be set to 6, etc.

Z is an optional parameter specifying the suppression of leading zeros.

, is an optional parameter for output which inserts commas between every thousands position of the value.

c The following five symbols are Credit Indicators which are optional parameters of the form:

C Causes the letters 'CR' to follow negative values and causes two blanks to follow positive or zero values.

D Causes the letters 'DB' to follow positive values; two blanks to follow negative or zero values.

M Causes a minus sign to follow negative values; a blank to follow positive or zero values.

E Causes negative values to be enclosed with a "<.....>" sequence; a blank follows positive or zero values.

N Causes the minus sign of negative values to be suppressed.

$ Is an optional parameter for output which appends a dollar

FORMAT MASK CODE:

#n specifies that the data is to be filled on a field of 'n' blanks.

*n specifies that the data is to be filled on a field of 'n' asterisks.

%n specifies that the data is to be filled on a field of 'n' zeros and to force leading zeros into a fixed field. 'D'() specifies the standard system 'D' (date) conversion.

NOTE: Any other character, including parentheses may be used as a field fill.

Explanation of the Format String Codes.

FORMAT:

variable = variable"{numeric mask code}{(format mask code)}"

NUMERIC MASK

MASK CODE	IMPLEMENTED AS	MEANING
j	R or L	Right or Left justification (default is left justification).
n	single numeric	# of decimal places.
m	single numeric	'Descaling' factor.
Z	Z	Suppress leading zeros.
,	,	Insert commas every thousands position.
c	C,D,M,E or N	Credit indicators.
$	$	Outputs dollar sign prior to value.

FORMAT MASK (enclosed in parentheses)

MASK CODE	IMPLEMENTED AS	MEANING
$	$	Outputs a dollar sign in the leftmost position of field.
#n	#10	Fills data on a field of 10 blanks.
%n	%10	Fills data on a field of 10 zeros.
*n	*10	Fills data on a field of 10 asterisks, or on a field of any other specified character.

NOTE: If a dollar sign is placed outside of the format mask, it will be output just prior to the value, regardless of the filled field. If a dollar sign is used within the format mask it will be output in the leftmost position regardless of the filled field.

General Form and Summary of Format String Codes.

UNCONVERTED STRING (X)	FORMAT STRING	RESULT (V)
X = 1000	V = X"R26"	10.00
X = 1234567	V = X"R27,"	1,234.57
X = -1234567	V = X"R27,E$"	$<1234.57>
X = 38.16	V = X"1"	38.2
X = -1234	V = X"R25$,M(*10#)"	***$123.40-
X = -1234	V = X"R25,M($*10#)"	$****123.40-
X = -1234	V = X"R25,M($*10)"	$***123.40-
X = 072458699	V = X"L(###-##-####)"	072-45-5866
X = 072458699	V = X"L(#3-#2-#4)"	072-45-5866
X = SMITH, JOHANNSEN	V = X"L((#13))"	(SMITH, JOHANN)

Sample usage of Numeric Format Codes.

The @ function generates terminal output codes to position the cursor to a specified location.

FORMAT:

```
@(column{,row})
```

where

column column at which to position cursor

row row at which to position the cursor; if row is not specified, cursor is positioned at column on current row

The values of the expressions used in the @ function must be within the row and column limits of the terminal screen. The upper left corner, also known as the home position, is location 0,0.

Special cursor control characters for the current terminal type (as defined by the TERM statement in effect at the time) can be generated by using negative values with @ function. For a list of values, see Table 9-1.

NOTE: The values generated by the @ function are specified in the TCL verb, DEFINE-TERMINAL.

Statement	Description
INPUT @(1,12) A	Cursor is at column 1, row 12; input prompt is displayed at column 0, row 12.
X = 7;Y = 3 print @(X,Y):Z	Prints value of variable Z at column 7, row 3
q = @(3):"HI" print q	Prints HI at column 3 of current line.
PRINT @(-1)	Clears the screen and positions the cursor at home location (0,0).

Table 9-1 Cursor Control Characters (1 of 2)

Character	Description
@(-1)	Clear screen and home cursor
@(-2)	Home cursor
@(-3)	Clear from cursor positon to the end of the screen
@(-4)	Clear from cursor position to the end of the line
@(-5)	Blink on
@(-6)	Blink off
@(-7)	Start protected field
@(-8)	Stop protected field
@(-9)	Backspace cursor one character
@(-10)	Move cursor up one line
@(-11)	Enable protect mode
@(-12)	Disable protect mode
@(-13)	Reverse video on
@(-14)	Reverse video off
@(-15)	Underline on
@(-16)	Underline off
@(-17)	Slave on
@(-18)	Slave off
@(-19)	Move cursor right
@(-20)	Move cursor down
@(-21)	Graphics character set on
@(-22)	Graphics character set off
@(-23)	Keyboard lock
@(-24)	Keyboard unlock
@(-25)	Control character enable
@(-26)	Control character disable
@(-27)	Write status line
@(-28)	Erase status line
@(-29)	Initialize terminal mode
@(-30)	Download function keys
@(-31)	Non-embedded stand-out on
@(-32)	Non-embedded stand-out off
@(-33)	Background - white
@(-34)	Background - brown
@(-35)	Background - magenta
@(-36)	Background - red
@(-37)	Background - cyan
@(-38)	Background - green
@(-39)	Background - blue
@(-40)	Background - black
@(-41)	Foreground, full-intensity - white
@(-42)	Foreground, full-intensity - brown
@(-43)	Foreground, full-intensity - magenta
@(-44)	Foreground, full-intensity - red
@(-45)	Foreground, full-intensity - cyan
@(-46)	Foreground, full-intensity - green
@(-47)	Foreground, full-intensity - blue
@(-48)	Foreground, full-intensity - black

TABLE 9-1 Cursor Control Characters (2 of 2)

Character	Description
@(-49)- @(-56)	Unused and reserved
@(-57)	Foreground, half-intensity - white
@(-58)	Foreground, half-intensity - brown
@(-59)	Foreground, half-intensity - magenta
@(-60)	Foreground, half-intensity - red
@(-61)	Foreground, half-intensity - cyan
@(-62)	Foreground, half-intensity - green
@(-63)	Foreground, half-intensity - blue
@(-64)	Foreground, half-intensity - black
@(-65) - @(-88)	Unused and reserved
@(-89)	80 x 25 black/white mode (monochrome monitor)
@(-90) - @(-92)	Unused and reserved
@(-93)	80 x 25 color mode
@(-94)	80 x 25 black/white mode (color monitor)
@(-95) - @(-98)	Unused and reserved
@(-99)	Returns 1 if the terminal uses embedded attributes; returns 0 if terminal does not use embedded attributes; returns null if not set in DEFINE-TERMINAL
@(-100)	Half intensity *
@(-101)	Full intensity *
@(-102) - @(-300)	Unused and reserved

*not supported for memory-mapped monitors

9.16 ABORT STATEMENT : TERMINATION

The ABORT statement may appear anywhere in the program; it designates a logical termination of the program.

FORMAT:

```
ABORT {errmsg.id{,param, param, ...}}
```

Upon the execution of a ABORT statement, the PICK/BASIC program will terminate.

The ABORT statement may be placed anywhere within the PICK/BASIC program to indicate the end of one of several alternative paths of logic. The ABORT statement is similiar to the STOP statement except that the ABORT statement will terminate execution of any PROC which might have called the program containing the ABORT statement.

Like the STOP statement, the ABORT statement may optionally be followed by an error message id, and error message parameters separated by commas. The error message name is a reference to an item in the ERRMSG file. The parameters are variables or literals to be used within the error message format.

```
PRINT 'PLEASE ENTER FILE NAME':
INPUT FN
OPEN '', FN TO FFN ELSE ABORT 201, FN
```

This program requests a file name and attempts to open the file. If an incorrect file name is entered, the standard system error message 201 "xxx IS NOT A FILE" will be printed, and the program is terminated.

Sample usage of the ABORT statement.

9.17 ABS FUNCTION : ABSOLUTE NUMERIC VALUE

The ABS function returns an absolute value. An absolute value is an unsigned integer value.

FORMAT:

ABS(expression)

The ABS function generates the absolute numeric value of the expression.

An absolute value is the numerical value of a number without reference to its algebraic sign. The result looks positive, but it is in fact, unsigned. For example:

```
A = 100 ; B = 25
C = ABS(B-A)
```

These statements assign the value 75 to variable C. (An absolute value is conventionally written as |75|.)

STATEMENT	EXPLANATION
A = ABS(Q)	Assigns the absolute value of variable Q to variable A.
A = 600 B = ABS(A-1000)	Assigns the value 400 to variable B.
A = 3 B = -10 C = ABS(A+B)	Assigns the value 7 to variable C.

Sample Usage of the ABS Function.

9.18 ALPHA FUNCTION : ALPHABETIC STRING DETERMINATION

The ALPHA function returns a value of true (1) if the given expression evaluates to a alphabetic character or string.

FORMAT:

ALPHA(expression)

The ALPHA function tests the given expression for a alphabetic value. For example, if the expression evaluates to an alphabetic character or alphabetic string the ALPHA function will return a value of true (i.e., generating a value of 1).

Inversely, an expression evaluating to a number, numeric string, or any non-alpha character will cause the ALPHA function to return a value of false. Consider the following examples:

```
IF ALPHA(expression) THEN PRINT "ALPHABETIC DATA"
```

This statement will print the text "ALPHABETIC DATA" if the current value of variable "expression" is a letter or an alphabetic string.

In the case of a non-numeric, non-alphabetic character or string (#, ?, etc.) a value of false would be returned for both the ALPHA and NUM functions.

The empty string ('') is considered to be a numeric string, but not an alpha string.

(See: NUM)

STATEMENT	EXPLANATION
A1=ALPHA("ABC")	Assigns a value of 1 to variable A1.
A3=ALPHA("12C")	Assigns a value of 0 to variable A3.
IF ALPHA(I CAT J) THEN GOTO 5	Transfers control to statement 5 if current value of both variables I and J are letters or alphabetic strings.

Sample Usage of NOT, NUM and ALPHA Functions.

9.19 ASCII FUNCTION : FORMAT CONVERSION

The ASCII function converts a string value from EBCDIC to ASCII.

FORMAT:

ASCII(expression)

The string value of the expression is converted from EBCDIC to ASCII. For example:

A = ASCII(B)

Conversely, the EBCDIC function is available to convert string values from ASCII to EBCDIC.

(See: EBCDIC)

STATEMENT	EXPLANATION
READT X ELSE STOP Y = ASCII(X)	Reads a record from the magnetic tape unit and assigns its value to variable X. Assigns ASCII value of record to variable Y.

Sample Usage of the ASCII function.

9.20 ASSIGNMENT STATEMENT : ASSIGNING VARIABLE VALUES

The Simple Assignment statement is used to assign a value to a variable.

FORMAT:

variable = expression

The resultant value of the expression becomes the current value of the variable on the left side of the equality sign. The expression may be any legal PICK/BASIC expression. For example:

```
ABC = 500
X2 = (ABC+100)/2
```

The first statement will assign the value of 500 to the variable ABC. The second statement will asign the value 300 to the variable X2 (i.e., X2 = (ABC+100)/2 = (500+100)/2 = 600/2 = 300).

String values may also be assigned. For example:

```
VALUE = "THIS IS A STRING"
SUB = VALUE [6,2]
```

The first statement above assigns the string "THIS IS A STRING" to variable VALUE. The second statement assigns the string "IS" to variable SUB (i.e., assigns to SUB the 2 character sub-string starting at character position 6 of VALUE).

STATEMENT	EXPLANATION
X=5	Assigns 5 to X.
X=X+1	Increments X by 1.
ST="STRING"	Assigns the character string to ST.
ST1=ST[3,1]	Assigns sub-string "R" to ST1.
TABLE(I,J)=A(3)	Assigns matrix element from vector element.
A=(B=0)	Assigns 1 to A if "B=0" is true, assigns 0 to A if "B=0" is false.

Examples of Assignment Statements.

9.21 BREAK ON AND OFF : DEBUGGER INHIBITION

The BREAK statements enable or disable the Debugger function accordingly.

FORMAT:

```
BREAK ON
BREAK OFF
BREAK expression
```

These commands increment/decrement the break inhibit counter. Note that they are cummulative. If two BREAK OFFs are executed, two BREAK ONs must be executed to restore a breakable status.

If the expression form of the command is used, the break key is disabled when the expression evaluates to 0. The break key is enabled when the expression evaluates to non-zero.

(See: PICK/BASIC DEBUGGER)

9.22 CALL AND SUBROUTINE STATEMENTS : EXTERNAL SUBROUTINES

The CALL and SUBROUTINE statements provide external subroutine capabilities for the PICK/BASIC program. An external subroutine is a subroutine that is compiled and cataloged separately from the calling program.

FORMAT:

```
CALL name(argument list)
SUBROUTINE name(argument list)
```

The CALL statement transfers control to the cataloged subroutine 'name'. The CALL 'argument list' consists of zero or more expressions, separated by commas, that represent actual values passed to the subroutine. The SUBROUTINE 'argument list' consists of the same number of expressions, by which the subroutine references the values being passed to it.

The SUBROUTINE statement is used to identify the program as a subroutine and must be the first statement in the program.

There is no correspondence between variable names or labels in the calling program and the subroutine. The only information passed between the calling program and the subroutine are the arguments. A sample external subroutine that involves two arguments together with correctly formed CALL statements, is shown below.

CALL Statements	Subroutine ADD
CALL ADD(A,B,C)	SUBROUTINE ADD(X,Y,Z)
CALL ADD(A+2,F,X)	Z=X+Y
CALL ADD(3,495,Z)	RETURN

An external subroutine must contain a SUBROUTINE statement and a RETURN statement. GOSUB and RETURN may be used within the subroutine. When a RETURN is executed with no corresponding GOSUB, control passes to the statement following the corresponding CALL statement. If the subroutine's END statement, a STOP or CHAIN statement (see appropriate section of the manual) is executed, control never returns to the calling program. The CHAIN statement should not be used to chain from an external subroutine to another PICK/BASIC program.

STATEMENTS	EXPLANATION
CALL REVERSE(A,B) SUBROUTINE REVERSE(I,X)	Subroutine REVERSE has two arguments.
CALL REPORT SUBROUTINE REPORT	Subroutine REPORT has no parameters.
CALL DISPLAY(A,B,C) SUBROUTINE DISPLAY(I,J,K)	Subroutine DISPLAY accepts (and returns) three argument values.

Sample Usage of CALL and SUBROUTINE Statements.

9.22.1 ARRAY PASSING AND THE CALL STATEMENT

Arrays may be passed to external subroutines. External subroutines may be called indirectly.

PASSING ARRAYS TO EXTERNAL SUBROUTINES

FORMAT:

```
MAT variable
```

The 'variable' is the name of the array defined in the DIMENSION statement. The array must be dimensioned in both the calling program and the subroutine. Array dimensions may be different, as long as the total number of elements matches. Arrays are copied in row major order. Consider the following example:

```
Calling Program                 Subroutine
DIM X(10), Y(10)                SUBROUTINE COPY (MAT A)
CALL COPY (MAT X, MAT Y)        DIM A(10,2)
END                             PRINT A(2,5)
                                RETURN
                                END
```

In this subroutine the parameter passing facility is used to copy MAT X and MAT Y specified in the CALL statement of the calling program into MAT A of the subroutine. Printing A(2,5) in the subroutine is equivalent to printing Y(5) in the calling program.

INDIRECT FORM OF THE CALL STATEMENT

FORMAT:

```
CALL @name(argument list)
```

The 'name' is a variable containing the name of the cataloged subroutine to be called. The argument list performs the same function as in a direct call.

```
NAME = 'XSUB1'
CALL @NAME
NAME = 'XSUB2'
CALL @NAME
```

The first call invokes subroutine XSUB1. The second call invokes subroutine XSUB2.

STATEMENTS	EXPLANATION
DIM A(4,10),B(10,5) CALL REV(MAT A, MAT B)	Subroutine REV accepts two input array variables, one of size 40 and one of size 50 elements.
SUBROUTINE REV(MAT C, MAT B) DIM C(4,10), B(50)	

Examples of Array Parameters.

9.23 CASE STATEMENT : CONDITIONAL BRANCHING

The CASE statement provides conditional selection of a sequence of BASIC statements.

FORMAT:

```
BEGIN CASE
    CASE expression
    statements
    CASE expression
    statements
    .
END CASE
```

If the logical value of the first expression is true (i.e., non-zero), then the statement or sequence of statements that immediately follows is executed, and control passes to the next sequential statement following the END CASE statement. If the first expression is false (i.e., zero), then control passes to the next test expression, and so on. Consider the following example:

```
BEGIN CASE
    CASE A < 5
    PRINT 'A IS LESS THAN 5'
    CASE A < 10
    PRINT 'A IS GREATER THAN OR EQUAL TO 5 AND LESS THAN 10'
    CASE 1
    PRINT 'A IS GREATER THAN OR EQUAL TO 10'
END CASE
```

If A<5, then the first PRINT statement will be executed. If 5<=A<10, then the second PRINT statement will be executed. Otherwise, the third PRINT statement will be executed. (Note that a test expression of 1 means "always true.")

STATEMENTS	EXPLANATION

BEGIN CASE
  CASE A=0; GOTO 10
  CASE A<0; GOTO 20
  CASE 1; GOTO 30
END CASE
	Program control branches to the statement with label 10 if the value of A is zero; to 20 if A is negative; or to 30 if A is greater than zero.

BEGIN CASE
  CASE ST MATCHES "1A"
  MAT LET=1
  CASE ST MATCHES "1N"
  SGL=1; A.1(I)=ST
  CASE ST MATCHES "2N"
  DBL=1; A.2(J)=ST
  CASE ST MATCHES "3N"
  GOSUB 103
END CASE
	If ST is one letter, "1" is assigned to all LET elements and the entire CASE is ended. If ST is one number, "1" is assigned to SGL, ST is stored at element A.1(I), and the entire case is ended. If ST is two numbers, "1" is assigned to DBL, ST is stored at element A.2(J), and the entire case is ended. If ST is three numbers, subroutine 103 is executed.

Sample usage of the CASE statement.

9.24 CHAIN STATEMENT : INTERPROGRAM COMMUNICATION

The CHAIN statement allows a PICK/BASIC program to execute any valid TCL command, including the ability to pass values to a separately compiled PICK/BASIC program which is executed during the same terminal session.

FORMAT:

```
CHAIN "any tcl command"
```

The CHAIN statement causes the specified TCL command to be executed. The CHAIN statement may contain any valid Verb or PROC name in the user's Master Dictionary. Consider the following example:

```
CHAIN "RUN FILE1 PROGRAM1"
```

This statement causes the previously compiled program named PROGRAM1 in the file named FILE1 to be executed.

By using the 'I' option, the CHAIN statement allows values to be passed to the specifed program. This is possible since all PICK/BASIC programs which are executed during a single terminal session use the same data area. The variables in one program that are to be passed to another program must be in the same location. This is accomplished via use of the DIM statement. Consider, for example, the following two PICK/BASIC programs:

Program ABC in file BP

```
DIM A(1,1), B(2)
A=500
B(1)=1 B(2)=2
CHAIN "RUN BP XYZ (I)"
END
```

Program XYZ in file BP

```
DIM I(2), J(1,1)
PRINT I(1)
PRINT I(2)
PRINT J(1,1)
END
```

Program ABC causes program XYZ to be executed. Program XYZ, in turn, prints the values "500", "1", and "2". All dimensioned variables form a long vector in row major order, and on a the CHAIN are assigned left to right to chained program's dimensioned variables.

The user should note that control is never returned to the PICK/BASIC program originally executing the CHAIN statement (See EXECUTE Statement).

(see: Execute statement)

STATEMENT	EXPLANATION
CHAIN "RUN FN1 LAX (I)"	Causes the execution of program LAX in file FN1 and values are passed to program LAX.
CHAIN "LISTU"	Causes the execution of the LISTU SYSPROG PROC.
CHAIN "LIST FILE"	Causes the execution of the LIST ACCESS Verb.
CHAIN "RUN PROGRAMS ABC"	Causes the execution of program ABC in file PROGRAMS. Since I option is not used, values will not be passed to program ABC.

Sample usage of the CHAIN statement.

9.25 CHAR FUNCTION : FORMAT CONVERSION

The CHAR function converts a numeric value to its corresponding ASCII character.

FORMAT:

CHAR(expression)

The CHAR function converts the numeric value specified by the expression to its corresponding ASCII character string value. For example, the following statement assigns the string value for as Attribute Mark to the variable AM:

AM = CHAR(254)

Conversely, the SEQ function is available to convert the first character of a string value to its corresponding numeric decimal value.

NOTE: For a complete list of ASCII codes, refer to the Appendix.

STATEMENT	EXPLANATION
SM = CHAR(255)	Assigns the string value for a Segment Mark to variable SM.
X = 252 SVM = CHAR(X)	Assigns the string value for a Secondary Value Mark to variable SVM.

Sample Usage of the CHAR Function.

9.26 CLEAR STATEMENT : INITIALIZING VARIABLE VALUES

The CLEAR statement is used to initialize all variables to a value of zero.

FORMAT:

CLEAR

The CLEAR statement initializes all possible variables to zero (i.e., assigns the value 0 to all variables). The CLEAR statement may be used in the beginning of the program to initialize all variables to zero, or may be used anywhere within the program for re-initialization purposes. This statement should only be used on completely debugged programs.

STATEMENT	EXPLANATION
CLEAR	Assigns the value 0 to all possible variables.

Example of the CLEAR Statement.

9.27 CLEARFILE STATEMENT : DELETING DATA

The CLEARFILE statement is used to clear out the data section of a specified file.

FORMAT:

```
CLEARFILE {file.variable}
```

Upon execution of the CLEARFILE statement, the data section of the file which was previously assigned to the specified file.variable via an OPEN statement, will be emptied. The data in the file will be deleted, but the file itself will not be deleted. If the file.variable is omitted from the CLEARFILE statement, then the internal default variable is used (thus specifying the file most recently opened without a file.variable).

The dictionary section of file should not be cleared via a CLEARFILE statement. If CLEARFILE is performed on a dictionary, opened as a data file, then all items of the dictionary will be deleted except for "D" pointers. PICK/BASIC program will abort with an appropriate error message if the specified file has not been opened prior to the execution of the CLEARFILE statement.

STATEMENT	EXPLANATION
OPEN 'FN1' ELSE PRINT 'NO FN1';STOP READ I FROM 'I1' ELSE STOP CLEARFILE	Opens the data section of file FN1, reads item I1 and assigns value to variable I, and finally clears the data section of file FN1.
OPEN 'FILEA' TO A ELSE STOP 201,'FILEA' OPEN 'FILEB' TO B ELSE STOP 201, FILEB' CLEARFILE A CLEARFILE B	Clears the data sections of files FILEA AND FILEB.
OPEN 'ABC' ELSE PRINT 'NO FILE'; STOP READV Q FROM 'IB3',5 ELSE STOP IF Q = 'TEST' THEN CLEARFILE	Clears the data section of file ABC if the 5th attribute of the item with name IB3 has a string value of 'TEST'.

Sample usage of the CLEARFILE statement.

9.28 COL1() AND COL2() FUNCTIONS : STRING SEARCHING

The COL1() and COL2() functions return the numeric values of the column positions immediately preceding and immediately following the sub-string selected by the FIELD function.

FORMAT:

```
COL1()
COL2()
```

COL1() returns the numeric value of the column position immediately preceding the sub-string selected via the most recent FIELD function. For example:

```
B = FIELD("XXX.YYY.ZZZ.555",".",2)
BEFORE = COL1()
```

These statements assign the numeric value 4 to the variable BEFORE (i.e., the value "YYY" which is returned by the FIELD function is preceded in the original string by column position 4).

COL2() returns the numeric value of the column position immediately following the sub-string selected via the most recent FIELD function. COL2() returns zero if the sub-string is not found. For example:

```
B = FIELD("XXX.YYY.ZZZ.555",".",2)
AFTER = COL2()
```

These statements assign the numeric value 8 to the variable AFTER (i.e., the value "YYY" which is returned by the FIELD function is followed in the original string by column position 8).

(See: FIELD)

STATEMENT	EXPLANATION
Q = FIELD("ABCBA","B",2) R = COL1() S = COL2()	Assigns the string value "C" to variable Q, the numeric value 2 to variable R, and the numeric value 4 to variable S.

Sample Usage of the COL1() and COL2() Functions.

9.29 COMMON STATEMENT : VARIABLE SPACE ALLOCATION

The COMMON statement may be used to control the order in which space is allocated for the storage of variables, and for the passing of values between programs.

FORMAT:

COM{MON} variable {,variable}...

The purpose of the COMMON statement is to change the automatic allocation sequence that the compiler follows, so that more than one program may have specified variables in a pre-determined sequence.

In the absence of a COMMON statement, variables are allocated space in the order in which they appear in the program, with the additional restriction that arrays are allocated space after all simple variables. COMMON variables (including COMMON arrays) are allocated space before any other variables in the program. The COMMON statement must appear before any of the variables in the program are used.

The COMMON variable list may include simple variables, file variables and arrays. Arrays may be declared in a COMMON statement by specifiying the dimensions enclosed in parentheses, (e.g. COMMON A(10) declares an array "A" with 10 elements). Arrays that are declared in a COMMON statement should not be declared again by a DIMENSION statement. All variables in the program which do not appear in a COMMON statement are allocated space in the normal manner.

The COMMON statement may be used to share variables among ENTERed programs, or among main-line programs and subroutines. This ensures that all 'COMMON' variables refer to the same external stored values in different programs. For example:

```
COMMON X,Y,Z(5)
COMMON Q,R,S(5)
```

If the first statement is found in a main-line program and the second in a subroutine call it is ensured that the variables X and Q, Y and R, and the arrays Z and S share the same locations. NOTE: The second COMMON statement variables may be regarded as a mask over the first. What associates Q to X (R to Y and S to Z) is a matter of alignment. Thus if the second statement had been "COMMON Q(2),R(5)" then Q(1) would refer to the location where the value of X is stored and Q(2) would refer to the location where the value of Y is stored.

The COMMON statement differs from the argument list in a Subroutine Call in that the actual storage locations of Common variables are shared by the main-line program and its external subroutines; whereas the argument list in a Subroutine Call causes the values to be pushed on to the stack. The COMMON statement thereby affords a more efficient method of passing values.

A program being called must have the same or less COMMON space.

Item "MAINPROG"

```
COMMON A,B,C(10)          Variables A, B, and array C are
A = "NUMBER"              allocated space before any other
B = "SQUARE ROOT"         variables.
FOR I = 1 TO 10
 C(I) = SQRT(I)
NEXT I
CALL SUBPROG              Subroutine call to cataloged program
PRINT "DONE"              SUBPROG.
END
```

Item SUBPROG

```
COMMON X(2),Y(10)         The 2 elements of array X contain
PRINT X(1), X(2)          respectively, the values of A and
FOR J = 1 TO 10           B from the main-line program.  The
 PRINT J, Y(J)            array Y contains the values of C
NEXT J                    from the main-line program.
RETURN                    Returns to main-line program.
END
```

Sample usage of the COMMON statement.

9.30 COS FUNCTION : COSINE OF AN ANGLE

The COS function generates the trigonmetric cosine of an angle.

FORMAT:

COS(expression)

The COS function generates the cosine of an angle, expressed in degrees.

Values which are less than 0 degrees, or greater than 360 degrees are adjusted to this range before generation.

(See: SIN)

STATEMENT	EXPLANATION
YY = COS(XX)	Assigns the cosine of an angle of XX degrees to YY.
PRINT COS(1)	Prints "0.9998"
PRINT COS(361)	Prints "0.9998"
PRINT COS(2)	Prints "0.9994"
PRINT COS(362)	Prints "0.9994"
PRINT COS(45)	Prints "0.7071"
PRINT COS(90)	Prints "0"

Sample usage of the COS function.

9.31 COUNT FUNCTION : DYNAMIC ARRAYS

The COUNT function counts the number of occurrences of a substring within a string.

FORMAT:

COUNT(string,substring)

The COUNT function counts the number of occurrences of a substring within a string. Any number of characters may be present in the substring. This function is particularly useful for determing the number of attributes within an item, or the number of multiple values or sub-values within an attribute (See DCOUNT).

The COUNT function returns a value of zero if the substring is not found, and returns the number of characters in the string if the substring is null (i.e. a null matches on any character). For example:

COMMAND	VALUE OF X
X = COUNT('THIS IS A TEST','IS')	2
X = COUNT('THIS IS A TEST','X')	0
X = COUNT('THIS IS A TEST','')	14

(There are 14 characters in the string.)

```
X = COUNT('AAAA','AA')                3
```

There are 3 substrings within the string AAAA.

```
AAAA          STRING
XX            SUBSTRING 1
 XX           SUBSTRING 2
  XX          SUBSTRING 3
```

(See: DCOUNT)

STATEMENT	EXPLANATION
A = "1234ABC5723" X = COUNT(A,'23')	Value returned in X is 2 as there are two occurances of '23' in the string A.
X = COUNT('ABCDEFG','')	Value returned in X is 7 as a null substring will match any character.

Sample examples of the COUNT function.

9.32 CRT STATEMENT : Terminal Output

The CRT statement is used to direct output to the terminal.

FORMAT:

```
CRT print.list
```

CRT is similar to the PRINT statement, except the CRT statement is not affected by the PRINTER ON or the PRINTER OFF statements.

The print.list can include @ functions to position data, literals, and expressions. Commas can be used to align data to preset tab positions at columns 18, 36, 54, and 72. Colons can be used to print data continuously across the page.

(See: PRINT AND PRINTER ON statements)

STATEMENT	EXPLANATION
CRT 'Print on terminal'	Output is to the terminal.
PRINTER ON PRINT 'Send to printer' CRT 'Send to terminal' PRINTER OFF	Directs print output to printer; CRT output is still directed to terminal

Sample Usage of the CRT statement.

9.33 DATA STATEMENT : STACKING INPUT DATA

The DATA statement is used to store data for stacked input when using the CHAIN or EXECUTE statement.

FORMAT:

```
DATA expression{,expression ...}
```

Where 'expression' may be any valid combination of variables, literals, functions, etc. Each expression becomes the response to one input request from the CHAIN or EXECUTE process.

Each DATA statement will generate one line of stacked input. The lines of stacked input are then used in response to the input requests of other processes. The DATA statement may be used to store stacked input for ACCESS, TCL, PROCs, or other PICK/BASIC programs.

The following example illustrates the procedure to exit a PICK/BASIC program, sort-select a file and begin execution of a second PICK/BASIC program. The variable REF.DATE is passed to the second PICK/BASIC program.

Assuming that no stacked input is currently present:

```
DATA 'RUN BP PROG'; DATA REF.DATE.B CHAIN 'SSELECT FILE WITH DATE
"':REF.DATE:'" BY DATE'
```

The first statement stacks two values (e.g. 'RUN BP PROG' and 'REF-DATE'). The second statement causes an ACCESS statement to be executed. When the ACCESS processor has completed, the first value on the stack is the input to the TCL prompt, thus BP PROG begins execution. (Note that the stack is a First In First Out (FIFO) type.)

The second PICK/BASIC program (BP PROG) then performs the following:

```
INPUT REF.DATE
```

This instruction gets its input from the second value on the stack, i.e. the value of REF.DATE from the first PICK/BASIC program.

NOTE: The DATA statement must be processed before the CHAIN or EXECUTE statement!!

STATEMENT	EXPLANATION
DATA A	Stacks the values of A, B
DATA B	and C for subsequent input requests.
DATA C	Program 'TEST' may have three
CHAIN 'RUN BP TEST'	input requests which will be satisfied by the stacked input.
DATA 'RUN BP CHARGE-ACC'	This causes the TCL command 'RUN
DATA DATE	BP CHARGE-ACC' to be stored on
CHAIN 'SELECT ACC WITH AMT > "100"'	the stack. Control first exits to the ACCESS processor to perform the SELECT, after which the PICK/BASIC program is run with DATE as stacked input.

Sample usage of the DATA statement.

9.34 DATE() FUNCTION : DATE CAPABILITY

The DATE() function returns the current internal date.

FORMAT:

DATE()

The DATE() function returns the string value containing the internal date. The internal date is the number of days since December 31, 1967.

(See: TIME() and TIMEDATE() functions)

STATEMENT	EXPLANATION
Q = DATE()	Assigns string value of current internal date to variable Q.
PRINT DATE()	Prints the current date in the internal format.
WRITET DATE() ELSE STOP	Writes the string value of the current internal date onto a magnetic tape record.

Sample Usage of the DATE() function.

9.35 DCOUNT FUNCTION : DYNAMIC ARRAYS

The DCOUNT Function returns a value which is the number of values separated by a specified delimiter.

FORMAT:

DCOUNT(string,substring)

The DCOUNT function counts the number of values separated by a specified delimiter. The DCOUNT function differs from the COUNT function in that it returns the true number of values by the specified delimiter, rather than the number of occurances of the delimiter within the string. For example, considering the string:

A = ABC^DEF^GHI^JKL

COMMAND	VALUE OF X
X = COUNT(A,AM)	3
X = DCOUNT(A,AM)	4

The DCOUNT function may be used to count the number of attributes in an item, or the number of values (or subvalues) within an attribute. The DCOUNT function returns a value of zero when a null string is encountered.

(See: COUNT)

STATEMENT	EXPLANATION
AM = CHAR(254) A = "123^456^ABC" X = DCOUNT(A,AM)	Value returned in X is 3 as there are three values in the string separated by attribute marks.
VM = CHAR(253) A = "123]456^ABC]DEF]HIJ" X = DCOUNT(A,VM)	Value returned in X is 4 as there are four values in the string separated by value marks.
A = "ABCDEFG" X = DCOUNT(A,'')	Value returned in X is 0 as a null is specified as the delimiter.

Examples of DCOUNT function.

9.36 DELETE STATEMENT : DELETING ITEMS

The DELETE statement is used to delete a file item.

FORMAT:

DELETE {file.variable,} item-name

The DELETE statement deletes the item which is specified by the itemname and which is located in the file previously assigned to the specified file.variable via an OPEN statement. If the file.variable is omitted, then the internal default variable is used (thus specifying the file most recently opened without a file.variable).

No action is taken if a non-existent item is specified in the DELETE statement.

The user should note that the PICK/BASIC program will abort with an appropriate error message if the specified file has not been opened prior to the execution of the DELETE statement.

(See: DELETE Function)

STATEMENT	EXPLANATION
DELETE X,"XYZ"	Deletes item XYZ in the file opened and assigned to variable X.
Q="JOB" DELETE Q	Deletes item JOB in the file opened without a file variable.

Sample Usage of the DELETE Statement.

9.37 DELETE FUNCTION : DYNAMIC ARRAY DELETION

The DELETE function deletes an attribute, a value, or a secondary value from a string in 'item' format (called a dynamic array).

FORMAT:

DELETE(da.variable,att#{,value#,sub-value#})

The dynamic array used by this function is specified by the da.variable. Whether an attribute, a value, or a secondary value is deleted depends upon the values of the second, third, and fourth parameters. The att# specifies an attribute, the value# specifies a value, and the sub-value# specifies a secondary value. If the value# and sub-value# both have a value of 0, or are dropped, then an entire attribute is deleted. If the last three expressions are all non-zero, then a secondary value is deleted.

If a value is deleted the value mark associated with the value is also deleted. If an attribute is deleted the attribute mark associated with the attribute is also deleted. Consider the following example:

```
OPEN 'TEST' TO TEST ELSE STOP 201,'TEST'
READ X FROM TEST,'NAME' ELSE STOP 202,'NAME'
WRITE DELETE(X,2) ON TEST,'NAME'
```

These statements delete attribute 2 (and its associated delimiter) of item NAME in file TEST.

STATEMENT	EXPLANATION
Y = DELETE(X,3,2)	Deletes value 2 of attribute 3 of dynamic array X (and its associated delimiter), and assigns resultant dynamic array to Y.
A=1;B=2;C=3 DA = DELETE(DA,A,B,C-A)	D eletes secondary value 2 (and its associated delimiter) of value 2 of attribute 1 of dynamic array DA.
X = DELETE(X,7)	Deletes attribute 7 (and its associated delimiter) of dynamic array X.
PRINT DELETE(X,7,1)	Prints the dynamic array which results when value 1 of attribute 7 of dynamic array X is deleted.

Sample usage of the DELETE Function.

9.38 DIM STATEMENT : DIMENSIONING ARRAYS

Multiple valued variables are called arrays. Before arrays may be used in a PICK/BASIC program they must be dimensioned via a DIM statement.

FORMAT:

```
DIM variable (dimension1{,dimension2})
```

A variable with more than one value associated with it is called an array. Each value is called an element of the array, and the elements are ordered. Before an array may be used in a PICK/BASIC program, however, the maximum dimension(s) of the array must be specified for storage purposes. This is done via a DIM statement, wherein the dimensions of an array are declared with constant whole number, separated by commas. DIM statements must precede any array references, and are usually placed at the beginning of the program. (Arrays need only be dimensioned once throughout the entire program.) Several arrays may be dimensioned via a single DIM statement.

```
          -------
          |  3  |---- The first element of A has value 3
          -------
          |  8  |---- The second element of A has value 8
Array A:  -------
          |-20.3|---- The third element of A has value -20.3
          -------
          | ABC |---- The fourth element of A has string value "ABC"
          -------
```

The above example illustrates a one-dimensional array (called a vector). A two-dimensional array (called a matrix) is characterized by having rows and columns. For example:

```
                COL.1  COL.2  COL.3  COL.4
                ---------------------------
          Row 1 |  3  | XYZ |   A  | -8.2 |
                |-------------------------|
Array Z:  Row 2 |  8  | 3.1 |  500 | .333 |
                |-------------------------|
          Row 3 |  2  |  -5 | Q123 |  84  |
                ---------------------------
```

Any array element may be accessed by specifying its position in the array. This position is like an offset from the beginning of the array. In specifying an element, the user must have one offset or subscript for each dimension of the array. In Array A, element A(1) has a value of 3, while element A(3) has a value of "20.3". For a two-dimensional array (matrix) the first subscript specifies the row, while the second specifies the column. For example, in array Z above, element Z(1,1) has a value of 3, while element Z(2,3) has a value of 500.

DIM MATRIX(10,12)	Specifies 10 by 12 matrix named MATRIX.
DIM Q(10),R(10),S(10)	Specifies three vectors named Q, R, and S (each to contain 10 elements).
DIM M1(50,10),X(2)	Specifies 50 by 10 matrix named M1, and two-element vector named X.

Sample usage of the DIM statement.

9.39 DTX FUNCTION : DECIMAL to HEXADECIMAL CONVERSION

The DTX function converts a value from Decimal to Hexadecimal.

FORMAT:

DTX(expression)

The string value of the expression is converted from Decimal to Hexadecimal. For example:

B = DTX(A)

Conversely, the XTD function is available to convert string values from Hexadecimal to Decimal.

(See: XTD)

STATEMENT	EXPLANATION
H = DTX(D)	Assigns the Hexadecimal value of variable D to variable H.

Sample Usage of the DTX function.

9.40 EBCDIC FUNCTION : FORMAT CONVERSION

The EBCDIC function converts a string value from ASCII to EBCDIC.

FORMAT:

EBCDIC(expression)

The string value of the expression is converted from ASCII to EBCDIC. For example:

B = EBCDIC(A)

Conversely, the ASCII function is available to convert string values from EBCDIC to ASCII.

(See: ASCII)

STATEMENT	EXPLANATION
B = EBCDIC(A)	Assigns the EBCDIC value of variable A to variable B.

Sample Usage of the EBCDIC function.

9.41 ECHO ON AND OFF : TERMINAL DISPLAY

The ECHO statement enables or disables terminal output accordingly.

FORMAT:

```
ECHO ON
ECHO OFF
ECHO expression
```

These commands turn the system echo-back on or off. They may be used to suppress the echo back of terminal input.

If the expression form is used, terminal echo is inhibited when the expression evaluates to zero. Terminal echo is enabled when the expression evaluates to non-zero.

9.42 END STATEMENT

If the END statement is used, it must be the last statement of the PICK/BASIC program; it designates the physical end of the program. The STOP and ABORT statements may appear anywhere in the program; they designate a logical termination of the program.

FORMAT:

```
END
```

The END statement may appear as the very last statement in the BASIC program. It is used to specify the physical end of the sequence of statements comprising the program, and increases readability.

The END statement is also used to designate the physical end of alternative sequences of statements within the IF statement and within certain of the PICK/BASIC I/O Statements.

(See: IF..THEN, LOCATE, LOCK, READ for a discussion of this alternative use of the END statement.)

```
      A=500 ; B=750 ; C=235 ; D=1300
      REVENUE = A + B ; COST = C + D
      PROFIT = REVENUE - COST
      IF PROFIT > 1 THEN GOTO 10
      PRINT "ZERO PROFIT OR LOSS"
      STOP<-----------Logical end of program.
  10  PRINT "POSITIVE PROFIT"
      END <---------------------------- Physical end of program
```

Sample usage of the END Statement.

9.43 ENTER STATEMENT : INTERPROGRAM TRANSFERS

The ENTER statement permits transfer of control from one cataloged program to another cataloged program. The program that executes the ENTER statement must be executed via the cataloged verb in the user's MD.

FORMATS:

```
ENTER program-name
```

where program-name is the item-id of the program to be ENTERed and

```
ENTER @variable
```

where variable has been assigned the program name to be ENTERed.
All variables which are to be passed between programs must be declared in a COMMON declaration in all program segments that are to be ENTERed.

All other variables will be initialized upon ENTERing the program. It is permissible to ENTER a program that calls a subroutine, but it is illegal to ENTER a program from a subroutine.

STATEMENT	EXPLANATION
ENTER PROGRAM.1	Causes execution of the cataloged program "PROGRAM.1". Any COMMON variables will be passed to "PROGRAM.1".
N=2 PROG = "PROGRAM." : N ENTER @PROG	Causes execution of the cataloged program "PROGRAM.2". Any COMMON variables will be passed to "PROGRAM.2".

Sample usage of the ENTER statement.

9.44 EQUATE STATEMENT : VARIABLE ASSIGNMENT

The EQUATE statement allows one variable to be defined as the equivalent of another variable.

FORMAT:

EQU{ATE} variable TO equate-variable{,variable TO equate.variable..}

The variable must be a simple variable. The equate-variable may be a literal number, string, character or array element. The equate-variable may also be a CHAR function, however, the CHAR function is the only allowed function in an EQUATE statement. The EQUATE statement must appear before the first reference to the equate-variable.

The EQUATE Statement differs from the ASSIGNMENT Statement (where a variable is assigned a value via an equal sign) in that there is no storage location generated for the variable. The advantage this offers is that the value is compiled directly into the object-code item at compile time and does not need to be re-assigned every time the program is executed. The EQUATE Statement is therefore particularly useful under the following two conditions:

Where a constant is used frequently within a program, and therefore the program would read more clearly if the constant were given a symbolic name. In the example, "AM" is the commonly used symbol for "attribute mark", one of the standard data delimiters.

Where a MATREAD statement is used to read in an entire item from a file and disperse it into a dimensioned array. In this case, the EQUATE statement may be used to give symbolic names to the individual array elements which makes the program more meaningful. For example:

```
DIM ITEM(20)
EQUATE BIRTHDATE TO ITEM(1), SOC.SEC.NO. TO ITEM(2)
EQUATE SALARY TO ITEM(3)
```

in this case, the variables BIRTHDATE, SOC.SEC.NO. and SALARY are rendered equivalent to the first three elements of the array ITEM. These meaningful variables are then used in the remainder of the program.

STATEMENT	EXPLANATION
EQUATE PI TO 3.1416	Variable PI is compiled as the value 3.1416 at compile time.
EQUATE STARS TO "*****"	Variable STARS is compiled as the value of five asterisks at compile time.
EQUATE AM TO CHAR(254)	Variable AM is equivalent to the ASCII charater generated by the CHAR function.
EQUATE PART TO ITEM(3)	Variable PART is equivalent to element 3 of array ITEM.

Sample usage of the EQUATE statement.

9.45 EXECUTE STATEMENT : EXECUTING TCL COMMANDS

The EXECUTE statement is used to execute any TCL command and use the results of that command in later processing.

FORMAT:

EXECUTE expression {CAPTURING var1} {RETURNING var2}

The 'expression' parameter may be a complete TCL statement, a PROC, or a cataloged PICK/BASIC program. Any output from the executed command is captured in 'var1'. After execution, 'var2' will contain error message numbers.

The CAPTURING and RETURNING clauses are both optional.

When using both clauses, the CAPTURING clause should precede the RETURNING clause.

After execution of the 'expression', the data stack is reset and the PICK/BASIC program continues with the next statement.

9.45.1 INPUT - EXECUTE STATEMENT

Input is passed to the EXECUTE statement using the DATA statement, just like it is used with the CHAIN statement. The data stack is reset after the EXECUTE statement is completed.

(See: DATA and CHAIN)

9.45.2 OUTPUT - CAPTURING CLAUSE

Output from the executed command is captured by the calling program in the variable used with the CAPTURING clause (var2). When output is being re-directed to a variable in the calling program, carriage-return/line feed pairs are converted to attribute marks, and clear-screen sequences (to the terminal) are deleted.

9.45.3 OUTPUT - RETURNING CLAUSE

Output of error message numbers may be examined in two ways. Using the optional RETURNING clause, allows error message numbers to be assigned to a variable (var1). Each error message number is separated by a blank. Secondly, the SYSTEM() function may be used.

9.45.4 SELECT LISTS - EXECUTE STATEMENT

If a selected list is active when the EXECUTE statement is executed, that list is passed to the TCL command executed. A selected list may be passed back from the executed command to the PICK/BASIC program, if one is generated. The select list will be assigned to the default select variable for the next READNEXT statement, or to any variable by:

SELECT TO variable

Therefore it is possible to EXECUTE the SELECT verb, test for select list active using the SYSTEM(11) function, and then EXECUTE the SAVE-LIST verb. It is also possible to issue a SELECT verb from TCL, RUN a PICK/BASIC program which EXECUTEs a LIST verb, and then have the initial select list passed to the LIST verb. In order to pass a select list to the executed TCL command, it is necessary that the select list not be referenced by a READNEXT or SELECT statement.

9.45.5 WORK ENVIRONMENT CHANGES

Extra care should be taken when using the following commands with the EXECUTE statement. The original environment will NOT be restored after they are EXECUTEd. The PICK/BASIC program will resume with the next line of code, under the newly changed parameters.

A) TERM
B) SP-ASSIGN, SP-OPEN, SP-CLOSE, etc.
C) P (output supression)
D) CHARGES - work performed with EXECUTE not reflected.
E) T-ATT, T-DET (record size, etc.)

There are two verbs which upon EXECUTion, do not return to the PICK/BASIC program.

A) OFF
B) LOGTO

There is one verb which cannot be done from an EXECUTion:

A) EXEC

9.45.6 EXECUTE WORKSPACE

The EXECUTE process requires it's own dedicated workspace. These workspace frames are automatically taken from overflow, and maintained in a special EXECUTE workspace table. The very first time an EXECUTE statement is performed, the process may be delayed up to 30 seconds. Subsequent EXECUTE statements will proceed without delay.

9.46 EXP FUNCTION : EXPONENTIAL CAPABILITY

The EXPONENTIAL function generates the result of raising base 'e' to the power designated by the expression. (Base 'e' is 2.7183)

FORMAT:

EXP(expression)

The EXPONENTIAL function raises the number 'e' (2.7183) to the value of the expression. If the value of the expression at precision 4 is 24 or greater, the function returns a value of zero. The size of the maximum result is reduced accordingly if precision 5 or precision 6 has been declared.

The EXPONENTIAL function is the inverse of the NATURAL LOGARITHM (LN) function. (See: LN)

STATEMENT	EXPLANATION
YY = EXP(XX)	Assigns the result of raising base 'e' the power of the expression XX, to variable YY.
PRECISION 6 PRINT EXP(1)	Prints "2.7182"
PRINT EXP(-110+120)	Prints "2.3026"
PRINT 24 + EXP(1000)	Prints "30.9079"
PRINT EXP(10000)	Prints "9.2105"

Sample usage of the EXP function.

9.47 EXTRACT FUNCTION : DYNAMIC ARRAY EXTRACTION

The EXTRACT function returns an attribute, a value, or a secondary value from a string in 'item' format (called a dynamic array).

FORMAT:

```
EXTRACT(da.variable,att#{,value#,sub-value#})
                        or
da.variable<att#{,value#,sub-value#}>
```

the dynamic array used by this function is specified by the da.variable. Whether an attribute, a value, or a secondary value is extracted depends upon the values of the second, third, and fourth parameters. The att# specifies an attribute, the value# specifies an value, and the sub-value# specifies a secondary value. If the third and fourth parameters both have a value of 0, or have been dropped, then an entire attribute is extracted. If the sub-value# (only) has a value of 0, or been dropped, then a value is extracted. If the last three parameters are all non-zero, then a secondary value is extracted. Trailing zero value# or sub-value# mark counts are not required. Consider the following example:

```
OPEN 'TEST' TO TEST ELSE STOP 201,'TEST'
READ ITEM FROM TEST,'NAME' ELSE STOP 202,'NAME'
PRINT ITEM<,3,2>
```

These statements cause value 2 of attribute 3 of item NAME in file TEST to be printed. Consider the following example:

```
OPEN 'ACCOUNT' TO ACCOUNT ELSE STOP 201,'ACCOUNT'
READ ITEM1 FROM ACCOUNT, 'ITEM1' ELSE STOP 202,'ITEM1'
IF ITEM1<3,2,1>=25 THEN PRINT "MATCH"
```

These statements cause the message "MATCH" to be printed if secondary value 1 of value 2 of attribute 3 of item ITEM1 in file ACCOUNT is equal to 25.

STATEMENT		EXPLANATION
Y=EXTRACT(X,2,0,0) Y=X<2>	or	Assigns attribute 2 of dynamic array X to variable Y.
A=3 B=2 Q1=ARR<A,B,A+1>		Assigns secondary value 4 of value 2 of attribute 3 of dynamic array ARR to variable Q1.
IF B<3,2,1> >5 THEN PRINT MSG GOSUB 100 END		If secondary value 1 of value 2 of attribute 3 of dynamic array B is greater than 5, then the value of MSG is printed and a subroutine branch is made to statement 100.
PRINT D<25,2,0>		Prints value 2 of attribute 25 of dynamic array D.

Sample usage of the EXTRACT Function.

9.48 FIELD FUNCTION : STRING SEARCHING

The FIELD function returns a sub-string from a string by specifying a delimiter character.

FORMAT:

FIELD(expression,delimiter,occurence#)

The FIELD function takes the string value of the expression and searches for a sub-string delimited by the character specified by the delimiter. The occurence# specifies which occurrence of the sub-string is to be returned. If the occurence# has a value of 1, then the FIELD function will return the sub-string from the beginning of the string up to the first occurrence of of the delimiter. For example, the statement below assigns the string value of "XXX" to the variable A:

```
A = FIELD("XXX.YYY.ZZZ.555",".",1)
```

If the occurence# has a value of 2, then the sub-string delimited by the first and second occurrence of the specified delimiter character will be returned. A value of 3 for the occurence# will return the sub-string delimited by the second and third occurence of the specified delimiter character, and so on for higher values. For example, the statement below assigns the string value "ZZZ" to variable C:

```
C = FIELD("XXX.YYY.ZZZ.555",".",3)
```

(See: COL1() and COL2() Functions)

STATEMENT	EXPLANATION
T = "12345A6789A98765A" G = FIELD(T,"A",1)	Assigns the string value "12345" to variable G.
T = "12345A6789A98765A" G = FIELD(T,"A",3)	Assigns the string value "98765" to variable G.
X = "77ABCXX" Y = "$" Z = "ABC" IF FIELD(X,Y,2)= Z THEN STOP	The IF statement will cause the program to terminate (i.e., the value returned by the FIELD function is "ABC", which equals the value of Z, thus making the test condition true).

Sample usage of the FIELD statement.

9.49 FOOTING STATEMENT : PAGE OUTPUT FOOTINGS

The FOOTING statement causes the specified text string to be printed at the bottom of each page.

FORMAT:

FOOTING "text 'options' (text 'options')"

The first FOOTING statement executed will initialize the page parameters. Subsequently, the Footing literal data may be changed by a new FOOTING Statement, and the new Footing will be output when the end of the current page is reached.

The special Footing option characters listed below may be used as part of a FOOTING string expression. These special characters will be converted and printed as part of the Footing. Option characters are enclosed in single quotes. Consider, for example:

FOOTING "Copyright 1988 PICK SYSTEMS 'T' PAGE 'P'"

This statement will print a Footing consisting of: the words "Copyright 1988 PICK SYSTEMS", followed by the current time and date, followed by the word "PAGE", followed by the current page number. Page numbers are assigned in ascending order starting with page 1.

The footing literal data may be changed at any time in the PICK/BASIC program by another FOOTING statement; this change will take effect when the end of the current page is reached. The same set of special option characters are used in heading statements.

(See: HEADING)

HEADING OPTIONS	Character is Converted to:
P	Current page number right justified in a field of four
L	Carriage return/line feed
T	Current time and date
C	Centers the line
D	Current date
N	No stop at end of page
PN	Current page number left justified

Special Option Characters for FOOTING Statement.

STATEMENT	EXPLANATION
FOOTING "TIME & DATE: 'T'"	The text "TIME & DATE:" will be printed followed by the current time and date.
FOOTING "PAGE 'P'"	The text "PAGE" will be printed followed by the current page number.
FOOTING "'LTP'"	The following footing will be printed: return/line feed, the current time, date and page.

Sample Usage of FOOTING Statements.

The FOR and NEXT statements are used to specify the beginning and ending points of a program loop. A loop is a portion of a program written in such a way that it will execute repeatedly until some test condition is met.

A FOR and NEXT loop causes execution of a set of statements for successive values of a variable until a limiting value is encountered. Such values are specified by establishing: 1) an initial value for a variable, 2) a limiting value for the variable, and 3) an increment value to be added to the value of the variable at the end of each pass through the loop. When the limit is exceeded, program control proceeds to the following body of the program.

FORMAT:

```
FOR variable = expression TO expression {STEP expression}
.
A=I*75
.
NEXT variable
```

The expression preceding TO specifies the initial value of the variable, the expression following TO gives the limiting value, and the optional expression following STEP gives the increment. If STEP is omitted, the increment value is assumed to be +1. The initial value expression is evaluated only once (when the FOR statement is executed). The other two expressions are evaluated on each iteration of the loop.

The function of the NEXT statement is to return program control to the beginning of the loop after a new value of the variable has been computed. Note that the variable in the NEXT statement must be the same as the variable in the FOR statement.

As an example, consider the execution of the following statements:

```
150 FOR J=2 TO 11 STEP 3
160 PRINT J+5
170 NEXT J
```

Statement 150 sets the initial value of J to 2 and specifies that J thereafter will be incremented by 3 each time the loop is performed, until J exceeds the limiting value 11. Statement 160 prints out the current value of the expression J+5. Statement 170 assigned J its next value (i.e., J=2+3=5) and causes program control to return to statement 150. Statement 160 is again executed, and statement 170 again increments J and causes the program to loop back. This process continues with J being incremented by 3 after each pass through the loop. When J attains the limiting value of 11, statement 160 will again be executed and control will pass to 170. J will again be incremented (i.e., J=11+3=14), and since 14 is greater than the limiting value of 11, the program will "fall through" statement 150 and control will pass to the next sequential statement following statement 170.

STATEMENTS	EXPLANATION
FOR A=1 TO 2+X-Y NEXT A	Limiting value is current value of expression 2+X-Y; increment value is +1.
FOR K=10 TO 1 STEP -1 . . NEXT K	Increment value is -1 (i.e., variable K will decrement by a value -1 for each of 10 passes through the loop).
FOR VAR= 0 TO 1 STEP 1 . . NEXT VAR	Increment value is 1 (i.e., variable VAR will increment by a value of 1 for each of 11 passes through the loop).

Sample usage of the FOR...NEXT statement.

Optional condition clauses (WHILE and UNTIL) may be used in the FOR statement. FOR and NEXT loops may be "nested"; a nested loop is defined as a loop which is wholly contained within another loop.

EXTENDED FORMAT:

```
FOR variable = expression TO expression [STEP
                    expression][WHILE expression]

FOR variable = expression TO expression [STEP
                    expression][UNTIL expression]
```

The extended form of the FOR statement functions identically to the basic form, with the following additions.

If the WHILE clause is used, the specified expression will be evaluated for each iteration of the loop. If it evaluates to false (i.e., zero), then program control will pass to the statement immediately following the accompanying NEXT statement. If it evaluates to true (i.e., non-zero), the loop will re-iterate.

If the UNTIL clause is used, the specified expression will be evaluated for each iteration of the loop. If it evaluates to true (i.e., non-zero), then program control will pass to the statement immediately following the accompanying NEXT statement. If it evaluates to false (i.e., non-zero), the loop will re-iterate.

The following FOR and NEXT loop, for example, will execute until I=10 or until the statements within the loop cause variable A to exceed the value 100:

```
FOR I=1 TO 10 STEP .5 UNTIL A>100
 .
 .
 A = I*75
 .
NEXT I
```

FOR and NEXT loops contained within the range of other FOR and NEXT loops are called nested loops. For example:

```
FOR I=1 TO 10
  FOR J=1 TO 10
  PRINT B (I,J)
  NEXT J
NEXT I
```

The above statements illustrate a two-level nested loop. The inner loop will be executed ten times for each of ten passes through the outer loop, i.e., the statement PRINT B(I,J) will be executed 100 times, causing matrix B to be printed in the following order: B(1,1), B(1,2), B(1,3),..., B(1,10), B(2,1), B(2,2),..., B(10,10).

Loops may be nested any number of levels. However, a nested loop must be completely contained within the range of the outer loop (i.e., the ranges of the loops may not cross).

STATEMENT	EXPLANATION
ST="X" FOR B=1 TO 10 UNTIL ST="XXXXX" ST=ST CAT "X" NEXT B	Loop will execute 4 times (i.e., an "X" is added to the string value of variable ST until the string equals "XXXXX").
A=20 FOR J=1 TO 10 WHILE A<25 A=A+1 PRINT J,A NEXT J	Loop will execute 5 times (i.e., variable A reaches 25 before variable J reaches 10).
A=0 FOR J=1 TO 10 WHILE A<25 A=A+1 PRINT J,A NEXT J	Loop will execute 10 times (i.e., variable J reaches 10 before variable A reaches 25).

Sample usage of the FOR...NEXT statement.
(Extended Form)

9.51 GOSUB AND ON...GOSUB STATEMENTS : SUBROUTINE BRANCHING

The GOSUB, COMPUTED GOSUB, RETURN, and RETURN TO statements (The RETURN and RETURN TO statements will be discussed in a following section.) provide internal subroutine capabilities for the PICK/BASIC program. A subroutine is an integral group of statements which handle a unique function or task. An internal subroutine is a subroutine that is contained within the program that calls it (i.e., before the END statement). The GOSUB statement transfers control to the subroutine).

FORMAT:

GOSUB statement.label

Upon execution of a GOSUB statement, program control is transferred to the statement which begins with the specified numeric statement.label. Execution proceeds sequentially from that statement until a RETURN or RETURN TO statement is encountered. Either of these statements transfers control back to the main program.

The Computed GOSUB statement is a combination of the Computed GOTO statement and the GOSUB statement. Control is transferred to one of several statement.labels selected by the current value of an index. expression. Control returns to the statement following the computed GOSUB when a RETURN statement is executed.

FORMAT: ON index. expression GOSUB statement.label, statement.label, ...

The index expression is evaluated and truncated to an integer value. The result is used as an index into the list of statement.labels. A subroutine branch is executed to the statement.label selected.

If the expression evaluates to less than 1 or to a value greater than the number of statement.labels, no action is taken, that is, the statement immediately following the ON GOSUB will be executed next.

```
        ON I GOSUB 100,150,250
        * CONTROL TRANSFERS HERE AFTER RETURN FROM SUBROUTINE
        . . .                                (DIRECTLY IF I<1 OR I>3)
100     * CONTROL TRANSFERS HERE IF I=1
        . . .
        RETURN
150     * CONTROL TRANSFERS HERE IF I=2
        . . .
        RETURN
250:    * CONTROL TRANSFERS HERE IF I=3
        . . .
        RETURN
```

Sample usage of the ON...GOSUB statement.

The GO{TO} statement unconditionally transfers program control to any statement within the PICK/BASIC program.

FORMAT:

GO{TO} statement-label

Execution of the GO{TO} statement causes program control to transfer to the statement which begins with the specified numeric statement-label. If a statement does not exist with the specified statement-label an error message will be printed at compile time (refer to the appendix describing compiler error messages). Note that control may be transferred to statements following the GO{TO} statement, as well as to statements preceding the GO{TO} statement.
(see: ON...GOTO)

```
----> 100 A=0
|          .
|          .
|          REM BRANCH TO STATEMENT 500
|      200 GOTO 500 --
|          .          |
|          .          |
|  ------------------
|  |       .
|  --> 500 A=B+C
|          D=100
|          .
|          .
|          REM REPEAT PROGRAM
|          GOTO 100 --
|-------------------|
```

Sample usage of the GOTO statement.

9.53 HEADING STATEMENT : PAGE OUTPUT HEADINGS

The HEADING statement causes the specified text string to be printed as the next page heading.

FORMAT:

HEADING "text 'options' {text 'options'}"

The first HEADING statement executed will initialize the page parameters. Subsequently, the Heading literal data may be changed by a new HEADING Statement, and the new Heading will be output at the beginning of the next page. The special heading option characters listed below may be used as part of a HEADING string expression. These special characters will be converted and printed as part of the heading. Option characters are enclosed in single quotes. Consider, for example:

HEADING "INVENTORY LIST 'T' PAGE 'PL'"

This statement prints a heading consisting of: the words "INVENTORY LIST", followed by the current time and date, followed by the word "PAGE", followed by the current page number, followed by a carriage return and line feed. Page numbers are assigned in ascending order starting with page 1.

The same set of special option characters are used in FOOTING statements.

(See: FOOTING)

HEADING OPTIONS	Character is Converted to:
P	Current page number right justified in a field of four
L	Carriage return/line feed
T	Current time and date
C	Centers the line
D	Current date
N	No stop at end of page
PN	Current page number left justified

Special Option Characters for HEADING Statement.

STATEMENT	EXPLANATION
HEADING "TIME & DATE: 'TL'	The text "TIME & DATE:" will be printed followed by the current time and date plus a carriage return/line feed.
HEADING "PAGE 'PL'"	The text "PAGE" will be printed followed by the current page number and a carriage return/line feed.

Sample Usage of HEADING Statements.

9.54 ICONV FUNCTION : INPUT CONVERSION

The ICONV function provides the PICK/ACCESS input conversion capabilities to the PICK/BASIC programmer.

FORMAT:

ICONV(expression,conversion)

The conversion specifies the type of input conversion to be applied to the string value resulting from the expression. The resultant value is always a string.

(See: OCONV)

The input conversion operation specified by the conversion parameter may include any one of the following:

D Convert date to internal format (for ICONV function) or to external format (for OCONV function).

MT Converts time.

MX Convert ASCII to hexadecimal (for ICONV function) or convert hexadecimal to ASCII (for OCONV function).

T Convert by table translation.

U Call to user-defined assembly routine.

For a detailed treatment of these (and other) conversion capabilities, the user should refer to the ACCESS Chapter.

NOTE: The ACCESS 'F' and 'A' conversions cannot be called by these functions. The ACCESS 'MR' or 'ML' conversion may be called by using the Format String which performs the same function and is preferable to using the ICONV or OCONV functions in this case.

STATEMENT	EXPLANATION
IDATE = ICONV("7-01-74","D")	Assigns the string value "2374" (i.e., the internal date) to the variable IDATE.
ITIME = ICONV("17:04:18","MT")	Assigns the string value "61458" (i.e., the internal time) to the variable ITIME.

Sample usage of the ICONV Function.

9.55 IF STATEMENT : SINGLE-LINE CONDITIONAL BRANCHING

The Single-Line IF statement provides the conditional execution of a sequence of PICK/BASIC statements, or the conditional execution of one of two sequences of statements.

FORMAT:

IF expression THEN statements {ELSE statements}

If the result of the test condition specified by the expression is true (i.e., non-zero), then the statement or sequence of statements following the THEN are executed. If the result of the expression is false (i.e., zero), then the statement or sequence of statements following the ELSE are executed, unless the ELSE clause is omitted, in which case control will pass to the next sequential statement following the entire IF statement. The expression may be any legal BASIC expression.

The sequence of statements in the THEN or ELSE clauses may consist of one or more statements on the same line. If more than one statement is contained in either the THEN or ELSE clause, they must be separated by semicolons. Consider the example:

```
IF ITEM THEN PRINT X; X=X+1 ELSE PRINT X*5; GOTO 10
```

If the current value of ITEM is non-zero (i.e., true), then this statement will print the current value of X, add one to the current value of X, and then transfer control to the next sequential instruction in the program. If the value of ITEM is zero (i.e., false), then the value of X*5 will be printed and control will transfer to statement 10.

Any statements may appear in the THEN and ELSE clauses, including additional IF statements.

The THEN clause of an IF statement is optional if the ELSE clause is present. One or the other MUST be present. This allows IF statements with the format:

IF expression ELSE statements

STATEMENT	EXPLANATION
IF A="STRING" THEN PRINT "MATCH"	Prints "MATCH" if value of A is the string "STRING".
IF Q THEN PRINT A ELSE PRINT B; STOP	The value of A is printed if Q is non-zero. If Q=0, then the value of B is printed and the program is terminated.
IF A=B THEN STOP ELSE IF C THEN GOTO 20	Program is terminated if A=B; control is passed to statement 20 if A does not equal B and if C is non-zero.

Sample usage of the Single-Line IF statement.

9.56 IF STATEMENT : MULTI-LINE CONDITIONAL BRANCHING

The Multi-Line IF statement is functionally identical to the Single-Line IF statement. It provides the conditional execution of a sequence of PICK/BASIC statements, or the conditional execution of one of two sequences of statements. The statement sequences, however, may be placed on multiple program lines.

The Multi-Line IF statement is actually an extension of the Single-Line format. With this format, the statement sequences in the THEN and ELSE clauses may be placed on multiple program lines, with each sequence being terminated by an END. The general format of the Multi-Line IF statement takes on three forms as shown in Figure A.

In each of the three forms, the ELSE clause is optional and may be included or omitted as desired. Any statements may appear in the THEN and ELSE clauses.

```
FORM 1:     IF expression THEN
            statements
            .
            .
            .
            END ELSE statements

FORM 2:     IF expression THEN
            statements
            .
            .
            .
            END ELSE
            statements
            .
            .
            .
            END

FORM 3:     IF expression THEN statements ELSE
            .
            .
            .
            END

NOTE:       In each of the above forms,
            the ELSE clause is optional.
```

General form of the Multi-Line IF statement.

IF STATEMENTS	EXPLANATION

IF ABC=ITEM+5 THEN
  PRINT ABC
  STOP
END ELSE PRINT ITEM; GOTO 10
	The value of ABC is printed and the program terminates if ABC=ITEM+5; otherwise the value of ITEM is printed and control passes to statement 10.

IF VAL THEN
  PRINT MESSAGE
  PRINT VAL
  VAL=100
END
	If the value of VAL is non-zero then the value of MESSAGE is printed, the value of VAL is printed, and VAL is assigned a value of 100; otherwise control passes to the next statement following END.

10 IF S="XX" THEN PRINT "OK" ELSE
  PRINT "NO MATCH"
  PRINT S
  STOP
END
20 REM REST OF PROGRAM
	If the value of S is the string "XX" then the message "OK" is printed and control passes to statement 20; otherwise "NO MATCH" is printed, the value of S is printed, and the program terminates.

IF X>1 THEN
  PRINT X
  X=X+1
END ELSE
  PRINT "NOT GREATER"
  GOTO 75
END
	If X>1 the value of X is printed and then incremented, and control passes to the next statement following the second END: otherwise "NOT GREATER" is printed and control passes to statement 75.

Sample usage of the Multi-Line IF statement.

9.57 IN Statement - Single Character Input

The IN statement is used to accept a single character of input. No prompt is displayed; no <RETURN> is expected.

The syntax of the statement is

```
IN variable
```

The input is stored in the variable as an ASCII (decimal) code.

Statement	Description
PRINT 'Press Y to continue': IN R	If Y is entered, the ASCII code 89 is returned in R.

9.58 INCLUDE STATEMENT : INCLUDING OTHER PICK/BASIC PROGRAMS

The INCLUDE statement is used to include data from other PICK/BASIC programs which normally consist of COMMON blocks and EQUATE statements.

FORMAT:

INCLUDE {file-name} item-name

When the PICK/BASIC compiler encounters an INCLUDE statement, it will open the specified file, read the item, and compile it into the current program.

If the file name is omitted, the file containing the source-item used in the TCL statement would be assumed.

Normally one would use this statement to include COMMON blocks and EQUATE statements into a program. It would also be logical to have CRT format strings and similar simple executable statements in the INCLUDE.

There is no limit to the number of INCLUDEs in a program, but only five levels of nesting are allowed.

STATEMENT	EXPLANATION
INCLUDE BP COMMON.DATA	The program COMMON.DATA in BP file will be compiled into the program containing the INCLUDE statement.

Sample usage of the INCLUDE statement.

NOTE: The INCLUDE statement is not available on the PC-XT Version 2.0 or lower.

9.59 INDEX FUNCTION : SEARCHING FOR SUB-STRINGS

The INDEX function searches a string for the occurrence of a sub-string and returns the starting column position of that sub-string.

FORMAT:

INDEX(string.expression,substring,occurrence#)

The INDEX function takes the string value of the expression and searches for the sub-string specified by substring. The occurrence# specifies which occurrence of that sub-string is sought. The resultant numeric value of the INDEX function is the starting column position of the sub-string within the string. a value of 0 is returned if the sub-string is not found. If the substring is null then the occurrence# will be returned.
The user should note that no blank space may appear between "INDEX" and "(". This is true for all PICK/BASIC Intrinsic Functions.

STATEMENT	EXPLANATION
A = INDEX("ABCAB","A",2)	Assigns value of 4 to variable A (i.e., 2nd occurrence of "A" is at column position 4 of "ABCAB").
X = "1234ABC" Y = "ABC" IF INDEX(X,Y,1)=5 THEN GOTO 3	The IF statement will transfer control to statement 3 (i.e., "ABC" starts at column position 5 of "1234ABC" which makes the test condition in the IF statement true).
Q = INDEX("PROGRAM","S",5)	Assigns value of 0 to variable Q (i.e., "S" does not occur in "PROGRAM").
S = "X1XX1XX1XX" FOR I=1 TO INDEX(S,"1",3) . . NEXT I	The loop will execute 8 times (i.e., 3rd occurrence of "1" appears at column position 8 of the string named S).

Sample usage of the INDEX Function.

9.60 INPUT STATEMENT : TERMINAL INPUT

The INPUT statement is used to request input data from the user's terminal. The input statement can include the @ function to position the cursor, and format strings to verify input.

The syntax of the statement is

```
INPUT {@(col,row)} variable {,len}{:} {mask}
```

where

@(col,row)	@ function; when specified, the cursor is positioned at the specified location and the carriage return/line feed after input is suppressed (for more information on positioning the cursor, see the section on @ function).
variable	receives response; if the variable being used already has a value, and if @ function has been specified, the current value of the variable is displayed as the default at the specified cursor address. To accept the default, press <RETURN>.
len	maximum number of characters to be entered; as soon as the specified number of characters are entered, an automatic <RETURN> is entered and processing continues with the next statement.
:	suppresses carriage return/line feed after input has been completed; this can be used only if @ function has not been specified.
mask	format string; if mask is to be used, the @ function must also be specified; for more information on using masks, see next section.

When an input statement is executed, a prompt character is displayed, followed by the cursor. If the @ function is used, the prompt is displayed preceding the location specified by @. The prompt character can be specified using the PROMPT statement.

9.60.1 Using Masks with Input Statement

The mask is used to verify and reformat the actual entry of the data. Any format string as described in Section 9.15, Numeric Masks and Format Mask Codes, can be specified. The input is verified against the mask, and, if acceptable, is assigned to the variable. For example, if the mask contains a decimal digit specification and/or a scaling factor, then numeric checking is performed. If the mask contains a length specification (e.g., R#10), then length checking is performed. If the mask is a valid date mask, then a date verification is performed.

Data is input and verified according to the mask, then stripped of its output characteristics and stored in internal format. For example, if the statement is

```
INPUT @(20,10) SOC.SEC '%%%-%%-%%%%'
```

and the data entered is

```
423-15-6897
```

the variable SOC.SEC contains the value

```
423156897
```

If the data is entered as 423156897, it is redisplayed at the input location as

```
423-15-6897
```

If fewer digits are entered than specified, the value is zero-filled and redisplayed; however, it is stored with the same number of digits as were entered.

If an error condition is encountered, a message is printed at the bottom of the screen and the cursor returns to the input prompt.

Error checking can be added to the INPUT @ form of the statement by the statements INPUTTRAP and INPUTNULL. Messages can be displayed using the INPUTERR statement. For more information, see the description for each statement.

Statement	Description
INPUT VAR	Requests a value for variable VAR.
INPUT VAR,3	Performs an automatic <RETURN> as soon as three characters are input (the user may press <RETURN> to enter fewer than three characters).
INPUT @(1,10) DESC	Cursor is positioned at column 1, row 10 for input. (The prompt is displayed at column 0, row 10.)
INPUT @(25,2) INV.DATE 'D'	Date can be input in either the form mm-dd-yy (any non-numeric character can be used as the delimiter) or the form dd mon yyyy; the date is redisplayed in the form dd mon yyyy and stored in internal date format.
INPUT @(35,7) AMOUNT 'R2,'	Data must be numeric; it is redisplayed right-justified and with two decimal places.
INPUT @(20,14) NAME 'L#40'	Data can be any characters; length of input is verified.

9.61 INPUTERR - INPUTTRAP - INPUTNULL : INPUT FORMS

Some extended features of the INPUT fuction.

FORMAT:

```
INPUTERR expr
INPUTTRAP 'xx' GOTO n,n,n,n ...
INPUTTRAP 'xx' GOSUB n,n,n,n ...
INPUTNULL x
```

These are all support functions for the extended form of input statement. They allow the user to tailor the INPUT function to conform to local standards.

INPUTERR causes a message, specified by "expr", to be printed on the last line of the screen. This differs from an explicit PRINT statement in that it sets a flag indicating that a message has been printed. Thus, when the next valid entry is made the system will check the flag and clear the bottom line.

INPUTTRAP allows the user to set a trap for a particular character or characters. Each character in the string specification corresponds to a label in the GOTO or GOSUB clause. Thus, for example, if the statement INPUTTRAP '__X' GOTO 10,20 is executed, the subsequent entry of a '__' character will cause a branch to "10" and the entry of 'X' will cause a branch to "20". The GOSUB form of this expression will cause a subroutine call to be issued instead. Caution - the subroutine RETURN statement will cause a return to the statement following the INPUTTRAP statement - not the one following the INPUT statement.

The INPUTNULL statement allows the user to define a character which is to signify that whatever default value was present is to be replaced by the null string. Thus, if the statement INPUTNULL '_/' is executed, the subsequent entry of a '_/' character will cause a defaulted value to go to null. Note that the default character is '__'.

(See: INPUT)

INPUTERR 'INVALID DATA!'	Displays error message
INPUTTRAP '*/' GOTO 150,170	Causes branching if either '*' or '_/' is entered.
INPUTNULL '_@'	Causes the '_@' character to null defaults in INPUT statements.

Examples of INPUTERR, INPUTTRAP and INPUTNULL Statements.

9.62 INSERT FUNCTION : DYNAMIC ARRAY INSERTION

The INSERT function inserts an attribute, a value, or a secondary value into a string in 'item' format (called a dynamic array).

FORMAT:

```
INSERT(da.variable,att#{,value#,sub-value#,}{;}new.expression)
```

The dynamic array used by this function is specified by the da.variable. Whether an attribute, a value, or a secondary value is replaced depends upon the values of the second, third, and fourth parameters. The att# specifies an attribute, the value# specifies a value, and the sub-value# specifies a secondary value. If the value# and sub-value# both have a value of 0, (or dropped) then an entire attribute is replaced. If the sub-value# (only) has a value of 0, (or dropped) then a value is replaced. If the second, third, and fourth parameters are all non-zero, then a secondary value is replaced. The replacement value is specified by the new.expression. The semi-colon (;) is used whenever value# and/or sub-value# have been dropped and the new.expression is no longer the fifth parameter.

If the att#, value# or sub-value# of the INSERT function has a value of -1, then insertion after the last attribute, last value, or last secondary value (respectively) of the dynamic array is specified. For example:

```
OPEN 'FN1' TO FN1 ELSE STOP 201,'FN1'
READ B FROM FN1,'ITEMX' ELSE STOP 202,'ITEMX'
A = INSERT(B,-1;'EXAMPLE')
WRITE A ON FN1,'ITEMX'
```

These statements insert the string value "EXAMPLE" after the last attribute of item ITEMX in file FN1.

STATEMENTS	EXPLANATION
Y = INSERT(X3,2,0,"XYZ")	Inserts before value 2 of attribute 3 of dynamic array X the string value "XYZ" (thus creating a new value), and assigns the resultant dynamic array to variable Y.
NEW = "VALUE" TEMP = INSERT(TEMP,9,0,0,NEW)	Inserts before attribute 9 of dynamic array TEMP the string value "VALUE" (thus creating a new attribute).
Z = INSERT(W,5,1,1,"B")	Inserts the string value "B" before secondary value 1 of value 1 of attribute 5 in dynamic array W (thus creating a new secondary value), and assigns the resultant dynamic array to variable Z.

Sample usage of the INSERT Function.

9.63 INT FUNCTION : INTEGER NUMERIC VALUE

The INT function returns an integer value. An integer is a whole number.

FORMAT:

INT(expression)

The INT function returns the integer portion of the specified expression (i.e., the fractional portion of the expression is truncated). For example:

```
PRINT INT(5.37)
```

.B This statement causes the value 5 to be printed.

STATEMENT	EXPLANATION
A = INT(Q)	Assigns the integer value of variable Q to variable A.
A = 3.55 B = 3.6 C = INT(A+B)	Assigns the value 7 to variable C.
J = INT (5/3)	Assigns the value 1 to variable J.

Sample Usage of the INT Function.

9.64 LEN FUNCTION : GENERATING A LENGTH VALUE

The LEN function determines the length of a string.

FORMAT:

LEN(expression)

the LEN function returns the numeric value of the length of the string specified by the expression. For example:

```
A = "1234ABC"
B = LEN(A)
```

These statements assign the value of 7 to variable B.

STATEMENT	EXPLANATION
Q = LEN("123")	Assigns the value 3 to variable Q (i.e., the length of string "123").
X = "123" Y = "ABC" Z = LEN(X CAT Y)	Assigns the value 6 to variable Z.

Sample Usage of the LEN Function.

9.65 LN FUNCTION : NATURAL LOGARITHM

The NATURAL LOGARITHM function generates the natural logarithm of the expression. (Base 'e' is 2.7183)

FORMAT:

LN(expression)

The NATURAL LOGARITHM (LN) function generates the natural (base e) logarithm of the expression. If the value of the expression is less than or equal to zero, the LN function returns a value of zero. The upper range limit for the expression is 14,073,748,835 at precision 4.

The NATURAL LOGARITHM function is the inverse of the EXPONENTIAL function.

(See: EXP)

STATEMENT	EXPLANATION
YY = LN(XX)	Assigns the natural logarithm of expression XX to variable YY.
PRINT LN(-35+37)	Prints "0.6932"
PRINT LN(1000)	Prints "6.9079"
PRINT LN(10000)	Prints "9.2105"

Sample usage of the LN function.

The LOCATE statement may be used to find the index of an attribute, a value, or a secondary value within a dynamic array. The elements of the dynamic array may be specified as being in ascending or descending ASCII sequence, and sorted with either right or left justification. If the specified attribute, value, or secondary value is not present in the dynamic array in the proper sequence, an index value is returned which may be used in an INSERT statement to place the sought element into its proper location.

FORMAT:

```
LOCATE('string',item{,att#{,val#}};index#{;'sequence}) THEN/ELSE stmts
```

'String' is the element to be located in dynamic array 'item'. 'Index#' is the variable into which the index of 'string' is to be stored. 'Att#' and "val#" are optional parameters which restrict the scope of the search within 'item'. If neither parameter is present, 'string' is tested for equality with attributes in 'item', and 'index#' returns an attribute number. If 'att#' is present, 'string' is compared with values within the attribute specified by "att#" of "item", and "index#" returns a value number. If 'val#' is also present, the search is conducted for secondary values of the specified attribute and value of 'item', and 'index#' returns a secondary value number.

If 'sequence' has the value 'A' (or any string value beginning with 'A'), the elements of "item" are assumed to be sorted in ascending sequence. If "sequence" has the value "D" (or any string value beginning with "D"), the elements are assumed to be in descending sequence. All other values for 'sequence' are ignored.

If the first character of 'sequence' is 'A' or 'D', the second character determines the justification used when sorting the elements. If the second character is "R", right justification is used. For any other value, including null, left justification is used. If 'sequence' is not specified and the string is not found, the default will be to the last position.

SEQUENCE PARAMETERS

AL - ascending, left-justified	**DL - descending, left-justified**
AR - ascending, right-justified	**DR - descending, right-justified**

The LOCATE statement has an alternate form that allows starting other than at the begining of a field.

Note: This alternate form is NOT available on the PC-XT Version 2.0 or lower.

FORMAT:

```
LOCATE string IN item {<att#{,val#}>} {,start} {BY seq} SETTING result
THEN/ELSE stmts.
```

Att#, val# and start form the AVS triple. If both att# and val# are present, the start is the starting SVM for the search. If only att# is present, then start is the starting val# for the search. If both att# and val# are ommited, the start is the starting att# for the search.

STATEMENT:

```
LOCATE('55',ITEM,3,1;INDEX1;'AR') ELSE ITEM = INSERT(ITEM,3,1,INDEX1,'55')
```

EXPLANATION

The third attribute, first value of dynamic array 'ITEM' is searched for the numeric literal '55'. 'INDEX1' will return with the secondary value index if the numeric is found, and will return with the correct secondary value index if the numeric is not found. If it is not found, control passes to the ELSE clause which inserts the numeric into the correct position by virtue of the index contained in 'INDEX1'. The optional parameter 'AR' specifies ascending sequence and right justification.

Sample usage of the the LOCATE statement.

9.67 LOCK STATEMENT : SETTING EXECUTION LOCKS

The LOCK statement provides a file and execution lock capability for PICK/BASIC programs. The LOCK statement sets execution locks while the UNLOCK statement releases them.

FORMAT:

LOCK expression {THEN/ELSE statements}

The LOCK statement sets an execution lock so that when any other BASIC program attempts to set the same lock, then that program will either execute an alternate set of statements or will pause until the lock is released (via an UNLOCK statement) by the program which originally locked it.

Execution locks may be used as file locks to prevent multiple PICK/BASIC programs from updating the same files simultaneously. There are 64 execution locks numbered from 0 through 63.

Note: There are only 48 execution locks on the PC-XT Version 2.0 and lower.

The value of the expression specifies which execution lock is to be set. If the specified execution lock has already been set by another concurrently running program (and the ELSE clause is not used), then program execution will temporarily halt until the lock is released by the other program.

If the ELSE clause is used, then the statement(s) following the ELSE will be executed if the specified lock has already been set by another program. The statements in the THEN/ELSE clause may be placed on the same line separated by semicolons, or may be placed on multiple lines terminated by an END (i.e., the THEN/ELSE clause takes on the same format as the THEN/ELSE clause in the IF statement).

All execution locks set by a program will automatically be released upon termination of the program.

(See: UNLOCK)

STATEMENTS	EXPLANATION
LOCK 15 ELSE STOP	Sets execution lock 15 (if lock 15 is already set, program terminates.
LOCK 2	Sets execution lock 2.
LOCK 10 ELSE PRINT X; GOTO 5	Sets execution lock 10 (if lock 10 is already set, the value of X is printed and program branches to statement 5.)

Sample Usage of the LOCK Statement.

Program loops may be constructed via the use of the LOOP statement.

FORMAT:

LOOP (statements) WHILE expression DO (statements) REPEAT

LOOP (statements) UNTIL expression DO (statements) REPEAT

Execution of a LOOP statement proceeds as follows. First the statements (if any) following "LOOP" will be executed. Then the expression is evaluated. One of the following is then performed depending upon the form used:

- If the "WHILE" form is used, then the statements following "DO" (if any) will be executed and program control will loop back to the beginning of the loop if the expression evaluates to true (i.e., non-zero), or program control will proceed with the next sequential statement following "REPEAT" (i.e., control passes out of the loop) if the expression evaluates to false (i.e., zero).

- If the "UNTIL" form is used, then the statements following "DO" (if any) will be executed and program control will loop back to the beginning of the loop if the expression evaluates to false (i.e., zero), or program control will proceed with the next sequential statement following "REPEAT" (i.e., control passes out of the loop) if the expression evaluates to true (i.e., non-zero).

Statements used within the LOOP statement may be placed on one line separated by semicolons, or may be placed on multiple lines. Consider the following example:

```
LOOP UNTIL A=4 DO A=A+1; PRINT A REPEAT
```

Assuming that the value of variable A is 0 when the LOOP statement is first executed, this statement will print the sequential values of A from 1 through 4 (i.e., the loop will execute 4 times). As a further example, consider the statement:

```
LOOP X=X-10 WHILE X>40 DO PRINT X REPEAT
```

Assuming, for example, that the value of variable X is 100 when the above LOOP statement is first executed, this statement will print the values of X from 90 down through 50 in increments of -10 (i.e., the loop will execute 5 times).

STATEMENTS	EXPLANATION

J=0
LOOP
  PRINT J
  J=J+1
WHILE J<4 DO REPEAT
	Loop will execute 4 times (i.e., sequential values of variable J from 0 through 3 will be printed).

Q=6 LOOP Q=Q-1 WHILE Q DO PRINT Q REPEAT	Loop will execute 5 times (i.e., values of variable Q will be printed in the following order: 5, 4, 3, 2, and 1).
Q=6 LOOP PRINT Q WHILE Q DO Q=Q-1 REPEAT	Loop will execute 7 times (i.e., values of variable Q will be printed in the following order: 6, 5, 4, 3, 2, 1, and 0).

B=1
LOOP UNTIL B=6 DO
  B=B+1
  PRINT B
REPEAT
	Loop will execute 5 times (i.e., sequential values of variable B from 2 through 6 will be printed).

Sample usage of the LOOP statement.

9.69 MAT - ASSIGNMENT AND COPY : ASSIGNING ARRAY VALUES

MAT Assignment and Copy statements are used to assign values to each element in the array.

FORMAT: MAT variable = expression

The MAT Assignment statement is similar to the Simple Assignment statement. It assigns a single value to all elements in an array.

The resultant value of the expression (which may be any legal expression) is assigned to each element of the array. The array being assigned is specified by the "variable" parameter. The specified array must have been previously dimensioned via a DIM statement. The following statement, for example, assigns the current value of X+Y-3 to each element of array A:

```
MAT A = X+Y-3
```

FORMAT: MAT variable = MAT variable

The MAT Copy statement copies one array to another. The first element of the array on the right becomes the first element of the array on the left, the second element on the right becomes the second element on the left, and so forth. Each variable name must have been dimensioned, and the number of elements in the two arrays must match; if not, an error message occurs.

Arrays are copied in row major order, i.e., with the second subscript (column) varying first. Consider the following example:

Program Code	Resulting Array Values
DIM X(5,2), Y(10)	X(1,1) = Y(1) = 1
FOR I=1 TO 10	X(1,2) = Y(2) = 2
Y(I)=I	X(2,1) = Y(3) = 3
NEXT I	.
MAT X = MAT Y	.
	X(5,2) = Y(10) = 10

The program dimensions two arrays as both having ten elements (5x2=10), initializes array Y elements to the numbers 1 through 10, and copies array Y to array X, giving the array elements the indicated values.

STATEMENTS	EXPLANATION
MAT TABLE=1	Assigns a value of 1 to each element of array TABLE.
MAT XYZ=A+B/C	Assigns the expression value to each element of array XYZ.
DIM A(20), B(20) . MAT A = MAT B	Dimensions two vectors of equal length, and assigns to elements of A the values of corresponding elements of B.
DIM TAB1(10,10), TAB2(50,2) . . MAT TAB1 = MAT TAB2	Dimensions two arrays of the same number of elements (10x10=50x2), and copies TAB2 values to TAB1 in row major order.

Sample usage of the MAT Assignment and Copy statements.

9.70 MATREAD STATEMENT : MULTIPLE ATTRIBUTES

The MATREAD statement reads a file item and assigns the value of each attribute to consecutive vector elements.

FORMAT:

MATREAD array.var FROM {file.variable,} itemname THEN/ELSE statements

The MATREAD statement reads the file item specified by the itemname and assigns the string value of each attribute to consecutive elements of the vector specified by the array.variable. If the file.variable is used, the item will be read from the file previously assigned to that file.variable via an OPEN statement. If the file.variable is omitted, then the internal default variable is used (thus specifying the file most recently opened without a file.variable).

If a non-existent item is specified, then the statements following the ELSE will be executed. The statements in the THEN/ELSE clause may appear on one line separated by semicolons, or on multiple lines terminated by an END (i.e., the THEN/ELSE clause takes on the same format as the THEN/ELSE clause in the IF statement). If the item does not exist, the contents of the vector remain unchanged.

If the number of item attributes is less than the DIMensioned vector size, the trailing vector elements are assigned a null string. If the number of attributes in the item exceeds the DIMensioned vector size, the remaining attributes will be assigned to the last element of the array.

(See: MATREADU)

STATEMENT	EXPLANATION
DIM ITEM (20) OPEN 'LOG' TO F1 ELSE STOP MATREAD ITEM FROM F1, 'TEST' ELSE STOP	Reads the item named TEST from the data file named LOG and assigns the string value of each attribute to consecutive elements of vector ITEM, starting with first element.

Sample Usage of the MATREAD Statement.

9.71 MATREADU STATEMENT : GROUP LOCKS

MATREADU provides the facility to lock a group of items in a file prior to updating an item in the group. Using a group lock prevents updating of an item by two or more programs simultameouslly while still allowing multiple program access to the file.

FORMAT:

MATREADU variable FROM {file.var,} itemname THEN/ELSE statements

This statement functions identically to the MATREAD statement, but additionally locks the group of the file in which the item to be accessed falls.

(See: MATREAD)

A group lock will prevent:

1. Access of items in the locked group of other PICK/BASIC programs using the READU, READVU, and MATREADU statements.

2. Update by any other program of any item in the locked group.

3. Access of the group by the FILE-SAVE process.

The group will become unlocked when any item in that group is accessed by the process which has it locked, when the PICK/BASIC program is terminated, or a RELEASE statement unlocks the group. Items can be updated to the group without unlocking it by using the WRITEU, WRITEVU or MATWRITEU statements.

Other processes (as in 1,2,3 above) which encounter a group lock will be suspended until the group becomes unlocked.

The maximum number of groups which may be locked by all processes in the system is 64. If a process attempts to lock a group when 64 locks are already set, it will be suspended until some group is unlocked.

(See: MATWRITEU)

STATEMENTS	EXPLANATION
MATREADU T FROM XM, "N4" ELSE NULL	This example shows use of a null ELSE clause to lock the group regardless of whether the item is existent or not.

Sample Usage of the MATREADU statement.

9.72 MATREADU STATEMENT : LOCKED CLAUSE

The MATREADU statement may be used with a LOCKED clause allowing the execution of statements if the group to be accessed is found to be already locked by another program.

FORMAT:

MATREADU var FROM {file.var,} itemname LOCKED stmts THEN/ELSE stmts

This statement functions exactly like the MATREADU statement, unless the group to be accessed is found to be already locked by another program, from another line. If the group to be accessed is found to be already locked, then the statements which follow the LOCKED clause will be executed.

If the LOCKED clause is not included in the MATREADU statement, the program will wait until the group it is trying to access becomes unlocked, before proceeding with the THEN or the ELSE clause.

(See: MATREADU)

STATEMENTS	EXPLANATION

MATREADU ARRAY1 FROM FILE1, IT1 LOCKED
 	
  GOTO 77
END THEN
  GOSUB 10
END ELSE
  GOSUB 20
END
	If group containing item IT1 is found to be already locked, the program will go to label 77. If the item IT1 exists the program will go to label 10. If the item IT1 does not exist, the program will go to label 20.

Sample Usage of a LOCKED clause with a MATREADU statement.

9.73 MATWRITE STATEMENT : MULTIPLE ATTRIBUTES

The MATWRITE statement writes a file item with the contents of a vector.

FORMAT:

MATWRITE array.variable ON {file.variable,} itemname

The MATWRITE statement replaces the attributes of the item specified by the itemname with the string value of the consective elements of the vector named by the array.variable. If the file.variable is used, the item will be written in the file previously assigned to that file.variable via an open statement. If the file.variable is omitted, then the internal default variable is used. If the itemname specifies an item which does not exist, then a new item will be created. The number of attributes in the item is determined by the DIMensioned size of the vector.

(See: MATWRITEU, WRITE, and WRITEV)

STATEMENT	EXPLANATION
DIM ITEM (10) OPEN '', 'TEST' ELSE STOP FOR I=1 TO 10 ITEM(I)=I NEXT I MATWRITE ITEM ON "JUNK"	Writes an item named JUNK in the file named TEST. The item written will contain 10 attributes whose string values are 1 through 10.

Sample Usage of the MATWRITE Statement.

The MATWRITEU statement has the letter "U" appended to it to imply update. This command will not unlock the group locked by the program.

FORMAT:

MATWRITEU variable ON {file.variable,} itemname

This command executes similar to the MATWRITE statement with the following added functionality.

(See: MATWRITE, WRITEU, and WRITEVU)

This command will not unlock the group locked by the program. This varient is used primarily for master file updates when several transactions are being processed and an update of the master item is made following each transaction update.

If the group is not locked when the MATWRITEU statement is executed, the group will not be locked by the execution of the command.

STATEMENT	EXPLANATION
MATWRITEU ARRAY ON FILE.NAME,ID	Replaces the attributes of the item specified by ID (in the file opened and assigned to variable FILE.NAME) with the consecutive elements of vector ARRAY. Does not unlock the group.

Sample usage of the MATWRITEU statement.

9.75 NOT FUNCTION : LOGIC CAPABILITY

The NOT function returns a value of true (1) if the given expression evaluates to 0 and a value of false (0) if the expression evaluates to a non-zero quantity.

FORMAT:

NOT(expression)

The NOT function returns the logical inverse of the specified expression; it returns a value of true (i.e., generates a value of 1) if the expression evaluates to 0, and returns a value of false (i.e., generates a value of 0) if the expression evaluates to a non-zero quantity. The specified expression must evaluate to a numeric quantity or a numeric string. The following statement, for example, assigns the value 1 to the variable X:

```
X = NOT(0)
```

As a futher example, the following statements cause the value 0 to be printed:

```
A = 1
B = 5
PRINT NOT(A AND B)
```

STATEMENT	EXPLANATION
X=A AND NOT(B)	Assigns the value 1 to variable X if current value of variable A is 1 and current value of variable B is 0. Assigns a value of 0 to X otherwise.
IF NOT(X1)THEN STOP	Program terminates if current value of variable X1 is 0.
PRINT NOT(M) OR NOT(NUM(N))	Prints a value of 1 if current value of variable M is 0 or current value of variable N is a non-numeric string. Otherwise prints a zero.

Sample usage of the NOT Function.

9.76 NULL STATEMENT : NON-OPERATION

The NULL statement specifies a non-operation, and may be used anywhere in the program where a PICK/BASIC statement is required.

FORMAT:

```
...NULL...
```

The NULL statement is used in situations where a PICK/BASIC statement is required, but no operation or action is desired. Consider the following example:

```
IF X1 MATCHES "9N" THEN NULL ELSE GOTO 100
```

This statement will cause program control to branch to statement 100 if the current string value of variable X1 does not consist of 9 numeric characters. If the current string value of variable X1 does consist of 9 numeric characters, then no action will be taken and program control will proceed to the next sequential PICK/BASIC statement.

The NULL statement may be used anywhere in the PICK/BASIC program where a statement is required.

STATEMENT	EXPLANATION
10 NULL	This statement does not result in any operation or action; however, since it is preceded by a statement label (10) it may be used as a program entry point for GOTO or GOSUB stmts elsewhere in the program.
IF A=0 THEN NULL ELSE PRINT "A NON-ZERO" GOSUB 45 STOP END	If the current value of variable A is non-zero, then the sequence of statements following the ELSE will be executed. If A=0, no action is taken and control passes to the next sequential statement following the END.
READ A FROM "ABC" ELSE NULL	File item ABC is read and assigned to variable A. If ABC does not exist, no action is taken. (Refer to description of READ statement for further information).

Sample usage of the NULL statement.

9.77 NUM FUNCTION : NUMERIC STRING DETERMINATION

The NUM function returns a value of true (1) if the given expression evaluates to a number or a numeric string.

FORMAT:

NUM(expression)

The NUM function tests the given expression for a numeric value. For example, if the expression evaluates to a number or numeric string the NUM function will return a value of true (i.e., generating a value of 1).

Inversely, an expression evaluating to a letter or an alphabetic string will cause the NUM function to return a value of false (0). Consider the following example:

```
IF NUM(expression) THEN PRINT "NUMERIC DATA"
```

This statement will print the text "NUMERIC DATA" if the current value of variable "expression" is a number or a numeric string. In the case of a non-numeric, non-alphabetic character or string (#, ?, etc.) a value of false would be returned for both the NUM and ALPHA functions. The empty string ('') and the period (.) are considered to be a numeric string, but not an alphabetic string.

(See: ALPHA)

STATEMENT	EXPLANATION
A1=NUM(123)	Assigns a value of 1 to variable A1.
A2=NUM("123")	Assigns a value of 1 to variable A2.
A3=NUM("12C")	Assigns a value of 0 to variable A3.

Sample Usage of the NUM Function.

9.78 OCONV FUNCTION : OUTPUT CONVERSIONS

The OCONV function provides the PICK output conversion capabilities to the PICK/BASIC programmer.

FORMAT:

OCONV(expression,conversion)

The conversion specifies the type of output ACCESS conversion to be applied to the string value resulting from the expression. The resultant value is always a string.

(See: ICONV)

The output conversion operation specified by the conversion parameter may include any one of the following:

D Convert date to internal format (for ICONV function) or to external format (for OCONV function).

MT Converts time.

MX Convert ASCII to hexadecimal (for ICONV function) or convert hexadecimal to ASCII (for OCONV function).

T Convert by table translation.

U Call to user-defined assembly routine.

For a detailed treatment of these (and other) conversion capabilities, the user should refer to the ACCESS chapter.

NOTE: The ACCESS 'F' and 'A' conversions cannot be called by these functions. The ACCESS 'MR' or 'ML' conversion may be called by using the Format String which performs the same function and is preferable to using the ICONV or OCONV functions in this case.

STATEMENT	EXPLANATION
A = "2374" B = "D" XDATE = OCONV(A,B)	Assigns the string value "01 JUL 1974" (i.e., the external date) to the variable XDATE.
A = OCONV(0, 'U50BB') PRINT A END	Assigns the string value of the line number and user account name to A. "02 SYSPROG" is printed.
NUM.LINES = OCONV(0,'UF070') PRINT NUM.LINES	Returns the number of physical serial I/O ports.

Sample Usage of the OCONV Function.

9.79 ON...GOTO STATEMENT : COMPUTED BRANCHING

The ON GOTO statement transfers control to one of several statement-labels selected by the current value of an index expression.

FORMAT:

```
ON index. expression GOTO statement.label, statement.label,...
```

Upon execution of the ON GOTO statement, program control is transferred to the statement which begins with the numeric statement.label selected by the expression. Statement.labels in the list are numbered 1, 2, 3,.... In executing the ON GOTO statement, the expression is evaluated and then the result of the expression is truncated to an integer value.

Consider the following example:

```
ON I GOTO 50, 100, 150
.
.
50  . . .
.
100 . . .
.
150 . . .
.
```

The labels in the label list may precede or follow the ON GOTO statement. If the current value of variable I=1, control transfers to the first statement.label, i.e., the statement with label 50. If I=2, control transfers to the third statement.label, i.e., statement 150.

If the value of the expression evaluates to less than one or greater than the number of statement.labels, no action is taken, that is, the statement immediately following the ON GOTO will be executed next.

STATEMENT	EXPLANATION
ON M+N GOTO 40, 61, 5, 7	Transfer control to statement 40, 61, 5, or 7 depending on the value of M+N being 1, 2, 3, or 4 respectively.
ON C GOTO 25, 25, 20	Transfer control to statement 25 if C = 1 or 2, to statement 20 in all other cases.
IF A GE 1 AND A LE 3 THEN ON A GOTO 110, 120, 150 END	The IF statement assures that A is in range for the computed GOTO statement.

Sample usage of the ON...GOTO statement.

9.80 OPEN STATEMENT : OPENING I/O FILES

The OPEN statement is used to select a file for subsequent input, output, or update. Before a file can be accessed by a READ, WRITE, DELETE, CLEARFILE, MATREAD, MATWRITE, READV, or WRITEV etc. statement, it must be opened via an OPEN statement.

FORMAT:

OPEN {"DICT,"}, "expression" {TO variable} THEN/ELSE statements

The expression in the OPEN statement indicates the file name. If the first parameter is DICT, then the dictionary section of the file is opened. The word DICT must be explicitly supplied to open a dictionary level file. If the file is a multiple data file (that is, multiple data files associated with a single dictionary), to open one of the data sections the format: 'dictname,dataname' is used.

If the "TO variable" option is used, then the dictionary or data section of the file will be assigned to the specified variable for subsequent reference. If the "TO variable" option is omitted, then an internal default variable is generated; subsequent I/O statements not specifying a file variable will then automatically default to this file.

If the file indicated in the OPEN statement does not exist, then the statement or sequence of statements following the ELSE will be executed. The statements in the ELSE clause may be placed on the same line separated by semicolons, or may be placed on multiple lines terminated by an END (i.e., the ELSE clause takes on the same format as the ELSE clause in the IF statement).

There is no limit to the number of files that may be open at any given time.

STATEMENT	EXPLANATION
A='DICT' OPEN A, 'XYZ' TO B ELSE PRINT "NO XYZ" STOP END	Opens the dictionary portion of file XYZ and assigns it to variable B. If XYZ does not exist, the text "NO XYZ" is printed and the program terminates.
OPEN '','ABC,X' TO D5 ELSE STOP	Opens data section X of file ABC and assigns it to variable D5. If ABC,X does not exist, program terminates.
X='' Y='TEST1' Z='NO FILE' OPEN X, Y ELSE PRINT Z; GOTO 5	Opens data section of file TEST1 and assigns it to internal default variable. If TEST1 does not exist, "NO FILE" is printed and control passes to statement 5.

Sample usage of the OPEN statement.

9.81 OUT Statement - Single Character Output

The OUT statement is used to output a single character; the character is specified as a decimal code; the corresponding ASCII character is printed.

The syntax of the function is

```
OUT var
```

The variable contains the code; a literal may also be used. Any ASCII code may be specified, including non-printable characters.

Statement	Description
OUT 80	The upper case letter P is displayed.
A = 1104 OUT A	Numbers greater than 256 are adjusted modulo 256; in this example, the upper case letter P is displayed.
BELL = 7 OUT BELL	Causes terminal bell to beep.

9.82 PAGE STATEMENT : HEADING OUTPUT

The PAGE statement causes the current output device to page, and causes the heading specified by the most recent HEADING/FOOTING statement to be printed as a page heading/footing. The page number may optionally be reset by the PAGE statement.

FORMAT:

PAGE (expression)

The PAGE statement causes the current output device to page, and causes the heading specified by the most recent HEADING statement to be printed at the top of the page. the number of print lines per page is controlled by the current TERM command (see TERM - TCL section). if a Footing statement has also been used, the PAGE statement will cause the footing to be printed out at the bottom of the page. If only a footing is desired, a null heading should be assigned. Headings and/or footings must be assigned before the PAGE statement is encountered.

If the PAGE statement has the optional expression, the expression is evaluated and the resulting number becomes the next page number used. If a FOOTING is in effect at the time that the page number is changed, the footing will be printed with a page number one less than the evaluated expression!

STATEMENT	EXPLANATION
HEADING "ANNUAL STATISTICS" FOOTING "XYZ CORPORATION" PAGE	The PAGE statement will cause both the specified heading and footing to be printed out when the paging is executed.
PAGE 1	This statement will cause the current footing, if any, to print (with a page number of 0), and the current heading, if any, to print with a page number of 1.
PAGE X+Y	The current footing and heading will be output, and the page number set to the evaluated result of X+Y.

Sample Usage of PAGE Statement.

9.83 PRECISION DECLARATION : SELECTING NUMERIC PRECISION

The PRECISION declaration allows the user to select the degree of precision to which all values are calculated within a given program.

FORMAT:

```
PRECISION n
```

n is a number from 0-6.

Note: Only PRECISION n, where n is a number 0-4 is supported on the PC-XT Version 2.0 or lower.

The default precision value is 4, that is, all values are stored in an internal form with 4 fractional places, and all computations are performed to this degree of precision. The desired number of fractional digits may be specified by a PRECISION declaration within the range of 0-6.

Only one PRECISION declaration is allowed in a program. If more than one is encountered, a warning message is printed and the declaration is ignored.

Where external subroutines are used, the mainline program and all external subroutines must have the same PRECISION. If the precision is different between the calling program and the subroutine, a warning message will be printed.

Changing the precision changes the acceptable form of a number; a number is defined as having a maximum of "n" fractional digits, where "n" is the precision value. Thus, the value:

```
1234.567
```

Is a legal number if the precision is 3 or 4, but is not a legal number if the precision is 0, 1 or 2.

Setting a precision of zero implies that all values are treated as integers. The max number at precision 4 is 14,073,748,835.

STATEMENT	EXPLANATION
PRECISION 0 A = 3 B = A/2	All numeric values in the program will be treated as integers. The value returned for B will be 1, not 1.5.
PRECISION 1	All numeric values in the program will be calculated to one fractional digit.
PRECISION 2	All numeric values in the program will be calculated to two fractional digits.
PRECISION 6	All numeric values in the program will be calculated to six fractional digits.

Sample Usage of PRECISION Declaration.

9.84 PRINT STATEMENT : TERMINAL OR PRINTER OUTPUT

The PRINT statement outputs data to the device selected by the PRINTER statement. The PRINT ON option allows output to multiple print files.

FORMAT:

PRINT (ON expression) print-list

The PRINT statement without the ON option is used to output variable or literal values to the terminal or line printer, as previously selected by a PRINTER statement. The print-list may consist of a single expression, or a series of expressions, separated by commas or colons (these punctuation marks are used to denote output formatting; refer to the section Tabulation and Concatenation in PRINT Statement). The expressions may be any legal PICK/BASIC expressions. The following statement, for example, will print the current value of the expression X+Y:

```
PRINT X+Y
```

The PRINT ON statement (i.e., with the ON option) is used, when PRINTER ON is in effect, to output the print-list items to a numbered print file. This is usually done when building several reports at the same time, each having a different number. The expression following ON indicates the print file number, which may be from 0 to 254 (selected arbitrarily by the program). Consider the following example:

```
PRINT ON 1 A,B,C,D
PRINT ON 2 E,F,G,H
PRINT ON 3 X,Y,Z
```

These statements will generate 3 separate output listings, one containing A, B, C, and D values, one containing E, F, G, and H values, and the third containing X, Y and Z values.

When the ON expression is omitted, print file zero is used.

The HEADING/FOOTING statements affect only print file zero. Pagination must be handled by the program for print files other than zero. Lack of pagination will result in continuous printing across page boundaries.

When PRINTER OFF is in effect, both PRINT ON and PRINT operate identically, i.e., all output is to the terminal. The contents of all print files used by the program, including print file zero, will be output to the printer in sequence when a PRINTER CLOSE statement is given or on termination of the program.

STATEMENT	EXPLANATION
PRINTER ON PRINT X	Causes the value of X to be output to print file 0.
PRINTER ON PRINT ON 24 X	Causes the value of X to be output to print file 24.
N=50 PRINT ON N X,Y,Z	Outputs print-list to print file 50.
PRINTER ON PRINT ON 15 "100" PRINT ON 40 "100"	Causes the value 100 to be copied to both print file 15 and print file 40.
PRINTER ON PRINT A PRINT B	Print file 0 will contain the values of A and B.
PRINTER ON PRINT ON 10 F1,F2,F3 PRINT ON 20 M,N,P PRINT ON 10 F4,F5,F6	Print file 10 will contain the values of F1 through F6; print file 20 will contain the values M, N and P.

Sample usage of the PRINT statement.

9.85 PRINT STATEMENT : TABULATION AND CONCATENATION

The print-list of the PRINT statement may specify tabulation or concatenation when printing multiple items.

Output values may be aligned at tab positions across the output page by using commas to separate the print-list expressions. Tab positions are pre-set at every 18 character positions. Consider the following example:

```
PRINT (50*3)+2, A, "END"
```

Assuming that the current value of A is 37, this statement will print the values across the output page as follows:

```
152               37                END
```

Output values may be printed continuously across the output page by using colons to separate the print-list expressions. The following statement, for example, will cause the text message "THE VALUE OF A IS 5010" to be printed:

```
PRINT "THE VALUE OF A IS" :50:5+5
```

After the entire print-list has been printed, a carriage return and a line feed will be executed, unless the print-list ends with a colon. In that case the next value in the next PRINT statement will be printed on the same line as the very next character position. For example, these statements:

```
PRINT A:B,C,D:
PRINT E,F,G
```

will produce exactly the same output as this statement:

```
PRINT A:B,C,D:E,F,G
```

STATEMENT	EXPLANATION
PRINT A:B: PRINT C:D: PRINT E:F	Prints the current values of A, B, C, D, E, and F contiguously across the output page, each value concatenated to the next.
PRINT A=1	Prints 1 if "A=1" is true; prints 0 otherwise.
PRINT A*100,Z	Prints the value of A*100 starting at column position 1; prints the value of Z on the same line starting at column position 18 (i.e., 1st tab position).
PRINT	Prints an empty (blank) line.
PRINT "INPUT":	Prints the text "INPUT" and does not execute a carriage return or line feed.
PRINT "", B	Prints the value of B starting at column position 18 (i.e., 1st tab position).

Sample usage of the PRINT statement formatting.

9.86 PRINTER ON/OFF STATEMENTS : SELECTING OUTPUT DEVICE

The PRINTER statement selects either the user's terminal or the line printer for subsequent program output.

FORMAT:

```
PRINTER ON
PRINTER OFF
PRINTER CLOSE
```

The PRINTER ON statement directs program output data specified by subsequent PRINT, HEADING/FOOTING, or PAGE statements to be output to the line printer. The PRINTER OFF statement directs subsequent program output to the terminal.

Once executed, a PRINTER ON or PRINTER OFF statement will remain in effect until a new PRINTER ON or PRINTER OFF statement is executed. If a PRINTER ON statement has not been executed, output will be to the terminal.

When a PRINTER ON statement has been issued, subsequent output data (specified by PRINT, HEADING/FOOTING, of PAGE statements) are not immediately printed on the line printer (Unless immediate printing is forced via the system SP-ASSIGN I or N option, as described in the PICK Peripheral Manual). Rather, the data are stored in an intermediate buffer area and are automatically printed upon termination of program execution.

If the user's application requires that the data be printed on the line printer prior to program termination, they may issue a PRINTER CLOSE statement. The PRINTER CLOSE statement will cause all data currently stored in the intermediate buffer area to immediately be printed.

When a PRINTER OFF statement has been issued, subsequent output data are always printed at the user's terminal immediately upon execution of the PRINT, HEADING, or PAGE statements (i.e., the PRINTER CLOSE statement applies only to output data directed to the line printer).

STATEMENT	EXPLANATION
PRINTER ON PRINT A PRINTER CLOSE PRINTER OFF PRINT B	Causes the value of variable A to be immediately printed on the line printer, and thereafter causes the value of variable B to be printed at the user's terminal.
PRINTER ON PRINT A PRINTER OFF PRINT B PRINTER CLOSE	Causes the value of variable B to be immediately printed at the user's terminal, and thereafter causes the value of variable A to be printed on the line printer.

Sample usage of the PRINTER ON/OFF/CLOSE statements.

9.87 PROCREAD STATEMENT : READING DATA FROM A CALLING PROC

The PROCREAD statement is used to read a calling PROC's primary input buffer and assign it to a variable within the program.

FORMAT:

PROCREAD variable THEN/ELSE statements

The PROCREAD statement reads the primary input buffer of the PROC from which the PICK/BASIC program, containing the PROCREAD statement was called, and assigns it to the variable.

The THEN/ELSE clause takes on the same format as the THEN/ELSE clause in the IF statement. If the primary input buffer has been cleared (i.e. RI or RO) before the PICK/BASIC program is executed, the variable is assigned to null, and the THEN statements are executed.

If the PICK/BASIC program is not called from a PROC, (i.e. from TCL) the ELSE statements will be executed.

(See: PROC)

```
 STATEMENT                          EXPLANATION

          CALL.PROC                 A proc named 'CALL.PROC'
   001 PQ
   002 HRUN BP PROG1                Run PROG1
   003 P

          PROG1                     The called program.
    .
    .
PROCREAD PROC.BUFF THEN             PROC.BUFF contains 'CALL.PROC'
   PRINT PROC.BUFF                  Prints out 'CALL.PROC'
END ELSE
   PRINT "NOT RUN FROM PROC"
END
    .
    .
```

Sample Usage of the PROCREAD Statement.

9.88 PROCWRITE STATEMENT : WRITING DATA BACK TO A CALLING PROC

The PROCWRITE statement is used to write a variable in a program, back to the primary input buffer of a calling PROC.

FORMAT:

PROCWRITE variable

The PROCWRITE statement will write whatever data is assigned to the variable, back into the primary input buffer of the PROC, which originally called the PICK/BASIC program.

If the PICK/BASIC program is not called from a PROC, (i.e. from TCL) nothing happens.

(See: PROC)

```
 STATEMENT                          EXPLANATION

          CALL.PROC                 A proc named 'CALL.PROC'
    001 PQ
    002 HRUN BP PROG1               Run PROG1
    003 P
    004 DO                          Displays input buffer - 'XYZ'

          PROG1                     The called program.
    .
    .
VAR1 = 'XYZ'
PROCWRITE VAR1                      Writes 'XYZ' to PROC buffer.
    .
    .
```

Sample Usage of the PROCREAD Statement.

9.89 PROMPT STATEMENT : INPUT PROMPT CHARACTER

The PROMPT statement is used to select the "prompt character" which is printed at the terminal to prompt the user for input.

FORMAT:

PROMPT expression

The value of the expression becomes the prompt character. For example:

PROMPT ":"

This statement selects the character ":" as the prompt character for subsequent INPUT statements. If the value of the expression is a numeric value of more than 1 digit, or a string consisting of one character, only the most significant character will be used.

When a PROMPT statement has been executed, it will remain in effect until another PROMPT statement is executed. If a PROMPT statement has not been executed, the INPUT statement will use a question mark (?) as the prompt character (i.e., "?" is the default prompt character).

(See: INPUT)

STATEMENT	EXPLANATION
PROMPT "@"	Specifies that the '@'character will be used as a prompt character for subsequent INPUT statements.
PROMPT A	Specifies that the current value of A will be used as a prompt character.

Sample Usage of the PROMPT Statement.

9.90 PWR FUNCTION : RAISING BY A POWER

The PWR function raises an expression by the power parameter.

FORMAT:

PWR(expression,power) or expression^power

The POWER function raises the expression to the power denoted by the power parameter. If the power parameter is zero, the function will return the value one.

If the expression raised to the power denoted by the power parameter is greater than 14,073,748,835 at precision 4, the function will return unpredictable numbers. If the expression is zero and the power parameter is any number other than zero, the function will return a value of zero. If the POWER PARAMETER is 0, the function will return a value of 0.
Note: another way to express the PWR function is X_^Y where X is raised to the Y power.

STATEMENT	EXPLANATION
YY = PWR(XX,ZZ)	Assigns the result of raising XX by the power of ZZ to the variable YY.
PRINT PWR(3+4,10)	Prints "282475249"
PRINT 6 + PWR(2,4)	Prints "22"
PRINT PWR(0,5)	Prints "0"

Sample usage of the PWR function.

9.91 READ STATEMENT : ACCESSING FILE ITEMS

The READ statement reads a file item and assigns its value to a variable.

FORMAT:

READ variable FROM {file.variable,} itemname THEN/ELSE statements

The READ statement reads the file item specified by the itemname and assigns its string value to the variable as a dynamic array. The file.variable is optional and specifies the file variable. If the file.variable is used, the item will be read from the file previously assigned to that file.variable via an OPEN statement. If the file.variable is omitted, then the internal default variable is used (thus specifying the file most recently opened without a file.variable).

If the itemname specifies the name of an item which does not exist, then the statement or sequence of statements following the ELSE will be executed. The statements in the THEN/ELSE clause may appear on one line separated by semicolons, or on multiple lines terminated by an END (i.e., the THEN/ELSE clause takes on the same format as the THEN/ELSE clause in the IF statement).

The user should note that the PICK/BASIC program will abort with an appropriate error message if the specified file has not been opened prior to the execution of the READ statement.

STATEMENT	EXPLANATION
READ A1 FROM X,"ABC" ELSE PRINT "NOT ABC" GOTO 70 END	Reads item ABC from the file opened and assigned to file variable X, and assigns its value to variable A1. If ABC does not exist, the text "NOT ABC" is printed and control passes to statement 70.
A="TEST" B="1" READ X FROM C,(A CAT B) ELSE STOP	Reads item TEST1 from the file opened and assigned to file variable C, and assigns its value to variable X. Program terminates if TEST1 does not exist.
READ Z FROM "Q" ELSE PRINT X; STOP	Reads item Q from the file opened without a file variable and assigns its value to variable Z. Prints value of X and terminates program if Q does not exist.

Sample usage of the READ statement.

9.92 READNEXT STATEMENT : ACCESSING ITEM-IDS

The READNEXT statement reads the next Item-id from a selected list. If multiple files have been selected, which list is specified by select.variable.

FORMAT:

READNEXT variable {,vmc}{FROM select.variable} THEN/ELSE statements

The READNEXT statement reads the next Item-id and assigns its string value to the variable indicated. The Item-id is read from the list created by the most recent program SELECT statement or SELECT, SSELECT, or QSELECT command issued at the TCL level. If the list of Item-id's has been exhausted, or if no selection has been performed, the statements following the ELSE will be executed. The statements in the THEN/ELSE clause may be placed on the same line separated by semicolons, or may be placed on multiple lines terminated by an END (i.e., the THEN/ELSE clause takes on the same format as the THEN/ELSE clause in the IF statement).

READNEXT FORMATS:.B

READNEXT variable THEN/ELSE statements
.B This will read the next Item-id of the last file selected without a select.variable.

READNEXT variable,vmc THEN/ELSE statements
.B The 'vmc' is used for the value mark count to be obtained from the Exploding Sort (External SSELECT).

READNEXT variable FROM select.variable THEN/ELSE statements
.B Reads the next Item-id of the file (or variable) selected and assigned to the select.variable.

READNEXT variable,vmc FROM select.variable THEN/ELSE statements
.B This is a combination of the previous two forms.

READNEXT A FROM X ELSE STOP	Specifies the list selected and assigned to the select-variable X. Assigns the value of that list's next item-id to variable A. If item-id list exhausted (or if no SELECT, SSELECT or QSELECT executed), program will terminate.
READNEXT X2 ELSE PRINT "UNABLE" GOTO 50 END	Specifies the last list selected without a select-variable. Assigns the value of the next item-id to variable X2. If unable to read, "UNABLE" is printed and control transfers to statement 50.

Sample usage of the READNEXT statements.

9.93 READT STATEMENT : READING RECORDS FROM TAPE

The READT statement is used to read records from magnetic tape. The record length is specified by the T-ATT statement executed at the TCL level. (For information on T-ATT, see Chapter 4, TCL Verbs.)

The syntax of the statement is

```
READT variable {THEN/ELSE statements}
```

The record is read and its string value is assigned to the variable indicated. If the tape unit has not been attached, or if an End-of-File (EOF) mark is read, the statements following ELSE are executed.

To read the error conditions, see the SYSTEM function.

Statement	Description

READT B ELSE
  PRINT "NO"
  GOTO 5
END
	The next tape record is read and its value assigned to variable B. If EOF is read (or tape unit not attached), then NO is printed and control passes to statement 5.

9.94 READU AND READVU STATEMENTS : GROUP LOCKS

READU and READVU provide the facility to lock a group of items in a file prior to updating an item in the group. Using a group lock prevents updating of an item by two or more programs simultameouslly while still allowing multiple program access to the file.

FORMAT:

READU variable FROM {file.var,} itemname THEN/ELSE statements

READVU variable FROM {file.var,} itemname,att# THEN/ELSE statements

These statements function identically to the READ and READV statements, but additionally lock the group of the file in which the item to be accessed falls. (See: READ and READV)

A group lock will prevent:

1. Access of items in the locked group of other PICK/BASIC programs using the READU, READVU, and MATREADU statements.

2. Update by any other program of any item in the locked group.

3. Access of the group by the file-save process.

The group will become unlocked when any item in that group is updated by the process which has it locked, when the PICK/BASIC program is terminated, or a RELEASE statement unlocks the group. Items can be updated to the group without unlocking it by using the WRITEU, WRITEVU or MATWRITEU statements.

Other processes (as in 1,2,3 above) which encounter a group lock will be suspended until the group becomes unlocked.

The maximum number of groups which may be locked by all processes in the system is 64. If a process attempts to lock a group when 64 locks are already set, it will be suspended until some group is unlocked.
(See: Matreadu)

STATEMENTS	EXPLANATION
READU ITEM FROM INV, S5 ELSE GOSUB 4 END	Lock group of items containing item S5. Read S5 to variable ITEM or, if S5 is non-existent, execute the ELSE clause; in either case the group remains locked until one of its items is updated or a RELEASE unlocks the group.
READVU ATT FROM B, "REC", 6 ELSE STOP	Lock group of items containing item REC. Read attribute 6 to variable ATT or, if REC is non-existent execute the ELSE clause. The group remains locked as above.

Sample Usage of READU and READVU statements.

9.95 READU AND READVU STATEMENTS : LOCKED CLAUSE

READU and READVU may be used with a LOCKED clause allowing the execution of statements if the group to be accessed is found to be already locked by another program.

FORMAT:

```
READU var FROM {file.var,} itemname LOCKED stmts THEN/ELSE stmts

READVU var FROM {file.var,} itemname,att# LOCKED stmts THEN/ELSE stmts
```

Note: The LOCKED CLAUSE portion of the READU and READVU is NOT available on the PC-XT Version 2.0 or lower.

These statements function identically to the READU and READVU statements, unless the group to be accessed is found to be already locked by another program, from another line. If the group to be accessed is found to be already locked, then the statements which follow the LOCKED clause will be executed.

If the LOCKED clause is not included in the READU or READVU statement, the program will wait until the group it is trying to access becomes unlocked, before proceeding with the THEN or the ELSE clause.

(See: READU and READVU)

STATEMENTS	EXPLANATION
READU ITEM FROM CUST, 101 LOCKED	If group containing item 101 is found to be already locked,
GOTO 99	the program will go to label 99.
END THEN	If item 101 exists, the program
GOSUB 10	will go to label 10.
END ELSE	If item 101 does not exist, the
GOSUB 20	program will go to label 20.
END	

Sample Usage of a LOCKED clause with a READU statement.

9.96 READV STATEMENT : ACCESSING AN ATTRIBUTE

The READV statement is used to read a single attribute value from an item in a file.

FORMAT:

READV variable FROM {file.variable,} itemname,att# THEN/ELSE statements

The READV statement reads the attribute specified by att# (attribute number) from the item specified by the itemname, and assigns its string value to the variable.

The file.variable is optional and specifies the file variable; if it is used, the attribute will be read from the file previously assigned to that file.variable via an OPEN statement. If the file.variable is omitted, then the internal default variable is used (thus specifying the file most recently opened without a file.variable).

If a non-existent item is specified, the statement or sequence of statements following the ELSE will be executed. The statements in the THEN/ELSE clause may be placed on the same line separated by semicolons, or may be placed on multiple lines terminated by END (i.e., the THEN/ELSE clause takes on the same format as the THEN/ELSE clause in the IF statement).

The PICK/BASIC program will abort with an appropiate error message if the specified file has not been opened prior to the execution of the READV statement.

STATEMENT	EXPLANATION
READV X FROM A, "TEST",5 ELSE PRINT ERR GOTO 70 END	Reads 5th attribute of item TEST (in the file opened and assigned to variable A) and assigns value to variable X. If item TEST is non-existent, then value of ERR is printed and control passes to statement 70.

Sample usage of the READV statements.

9.97 RELEASE STATEMENT : RELEASING GROUP UPDATE LOCKS

The RELEASE statement unlocks specified groups or all groups locked by the program.

FORMAT:

RELEASE {{file.variable,} expression}

The RELEASE statement unlocks the group hashed into by the item-id specified by the expression. If the file.variable is used, the file will be the one previously assigned to that file.variable via on OPEN statement. If the file.variable is omitted, then the internal default variable is used (thus specifying the file most recently opened without a file.variable).

If the RELEASE statement is used without a file.variable or expression all groups which have been locked by the program will be unlocked.

The RELEASE statement is useful when an abnormal condition is encountered during multiple file updates. A typical sequence is to mark the item with an abnormal status, update it to the file and then RELEASE all other locked groups. This version of the RELEASE statement will release all groups locked by the program.

(See: READU, READVU and MATREADU)

STATEMENT	EXPLANATION
RELEASE	Releases all groups locked by the program.
RELEASE CUST.FILE, PART.NO	Releases group hashed into by item-id contained in PART.NO in file CUST.FILE.

Sample usage of the RELEASE statement.

9.98 REM OR MOD FUNCTION : REMAINDER VALUE

The REM or MOD function generates the remainder (modulo) of one number divided by another.

FORMAT:

REM(numerator,denominator)
or
MOD(numerator,denominator)

This function returns the remainder (modulo) of the value of the numerator divided by the value of the denominator.

The REM and MOD (modulo) functions are identical.

STATEMENT	EXPLANATION
A = MOD(Q,Z)	Assigns the remainder of variable Q divided by Z to variable A.
A = 600 B = REM(A,-1000)	Assigns the value 600 to variable B.
J = REM(5,3)	Assigns the value 2 to variable J.
Q = MOD (1023,256)	Assigns the value 255 to the variable Q.

Sample Usage of the REM or MOD function.

9.99 REPLACE FUNCTION : DYNAMIC ARRAY REPLACEMENT

The REPLACE function replaces an attribute, a value, or a secondary value in a string in 'item' format (called a dynamic array).

FORMAT:

```
REPLACE(da.variable,att#{,value#,sub-value#,}{;}new.expression)
                                or
  da.variable<att#{,value#,sub-value#}> = new.expression
```

The second form above is actually an extract function being utilized as a replacement function. The dynamic array used by this function is specified by the da.variable. Whether an attribute, a value, or a secondary value is replaced depends upon the values of the second, third, and fourth parameters. The att# specifies an attribute, the value# specifies a value, and the sub-value# specifies a secondary value. If the value# and sub-value# both have a value of 0, (or dropped) then an entire attribute is replaced. If the sub-value# (only) has a value of 0, (or dropped) then a value is replaced. If the second, third, and fourth parameters are all non-zero, then a secondary value is replaced. The replacement value is specified by the new.expression. The semi-colon (;) is used whenever value# and/or sub-value# have been dropped and the new.expression is no longer the fifth parameter.

If the att#, value# or sub-value# of the REPLACE function has a value of -1, then insertion after the last attribute, last value, or last secondary value (respectively) of the dynamic array is specified. For example:

```
OPEN 'XYZ' TO XYZ ELSE STOP 201,'XYZ'
READ B FROM XYZ,'ABC' ELSE STOP 202,'ABC'
B<3,-1>='NEW VALUE'
WRITE B ON XYZ,'ABC'
```

These statements insert the string value "NEW VALUE" after the last value of attribute 3 of item ABC in file XYZ.

STATEMENT	EXPLANATION
Y=REPLACE(X,4,0,0,'')	Replaces attribute 4 of dynamic array X with the empty (null) string. string, and assigns the resultant dynamic array to Y.
VALUE="TEST STRING" DA<4,3,2>=VALUE	Replaces secondary value 2 of value 3 of attribute 4 in dynamic array DA with the string value "TEST STRING".
X="ABC123" Y<1,1,-1>=X	Inserts the value "ABC123" after the last secondary value of value 1 of attribute 1 in dynamic array Y.

Sample usage of the REPLACE Function.

The RETURN or RETURN TO statements return control to the main program.

FORMAT:

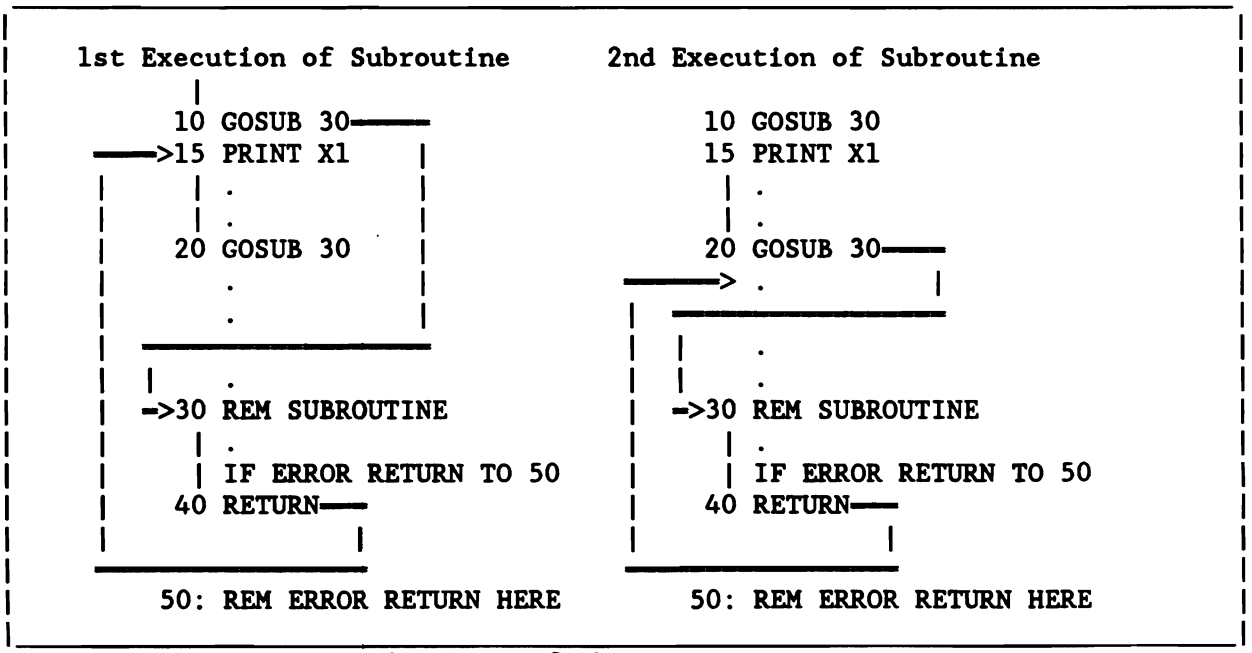
```
RETURN
RETURN TO statement-label
```

The RETURN statement will transfer control from the subroutine back to the statement immediately following the GOSUB statement. The RETURN TO statement returns control from the subroutine to the statement within the PICK/BASIC main program having the specified statement-label.

The statements in a subroutine may be any PICK/BASIC statements, including another GOSUB statement. To insure proper flow of control, each subroutine must return to the calling program by using a RETURN (or RETURN TO) statement, not a GOTO statement. The user should also insure that the subroutine cannot be executed by any flow of control other than through the execution of a GOSUB statement.

If the RETURN TO statement refers to a statement-label which is not present in the program, an error message will be printed at compile time (refer to APPENDIX C - PICK/BASIC COMPILER ERROR MESSAGES).

Consider the statements shown in the example below . Upon execution of statement 10, control will transfer to statement 30 as illustrated in the left side of the figure. The statements within the subroutine will be executed and statement 40 will then return control to statement 15. Execution will then proceed sequentially to statement 20, whereby control will again be transferred to the subroutine as shown in the right side of the figure. The conditional RETURN TO path is taken instead of the normal RETURN if the logical variable ERROR is true (1).



Sample usage of the RETURN statement.

9.101 REWIND STATEMENT : REWINDING THE TAPE

BASIC programs may specify Magnetic Tape to rewind to the BOT (Beginning of Tape) mark through the use of the REWIND (Rewind Tape Unit) statement.

FORMAT:

REWIND THEN/ELSE statements

The REWIND statement rewinds the magnetic tape unit to the Beginning-of-Tape (BOT). If the tape unit has not been attached, then the statement(s) following the ELSE will be executed.

STATEMENT	EXPLANATION
REWIND ELSE STOP	Tape is rewound to BOT.

Sample Usage of the REWIND statement.

9.102 RND FUNCTION : RANDOM NUMBER GENERATION

The RND function returns a random number. The range of the random number generated is controlled by the expression.

FORMAT:

RND(expression)

The RND function generates a numeric value for a random number between zero and the number specified by the expression less one (inclusive), which must be positive.

Therefore, an expression parameter which evaluates to 3, would randomly generate 0, 1, or 2. This is an invaluable function when programming games of chance.

STATEMENT	EXPLANATION
Z = RND(11)	Assigns a random number between 0 and 10 (inclusive) to the variable Z.
R = 100 Q = 50 B = RND(R+Q+1)	Assigns a random number between 0 and 150 (inclusive) to the variable B.
Y = RND(ABS(051))	Assigns a random number between 0 and 50 (inclusive) to the variable Y.

Sample Usage of the RND Function.

The SELECT command provides a facility to select a set of item-ids or attributes which, when used in conjunction with the READNEXT statement (see section on READNEXT), may be used to access single or multiple file item-ids or attributes within a PICK/BASIC program.

FORMAT:

SELECT {file.variable}{TO select.variable}

The SELECT statement builds the same list of item-ids as a SELECT command executed at the TCL level without any selection criteria (see ACCESS). If the file.variable is used, a list of item-ids will be created for the file or item previously assigned to that file.variable via an OPEN or READ statement. If the file.variable is omitted, then the internal default variable is used (thus specifying the file most recently opened without a file.variable). The item ids are then extracted using the READNEXT statement.

FORMAT

SELECT

Creates a select list of item-ids from the file most recently opened without a file.variable.

SELECT file.variable

Creates a select list of item-ids from the file opened to 'file.variable'.

SELECT var

Creates a select list from the attributes of the variable 'var'. The select list only includes the first value of a multivalued attribute.

SELECT TO select.variable

Creates a select list from the file most recently opened without a file. variable and assign the selected list to 'select.variable'.

SELECT file.variable TO select.variable

Creates a select list from the file opened to 'file.variable' and assign the selected list to 'select.variable'.

SELECT var TO select.variable

As above, except the selected list is assigned to 'select.variable'.

STATEMENT	EXPLANATION
SELECT	Builds list of item-id's using the default variable of the last file opened without a file-variable.
SELECT BP TO BLIST	Builds a list of item-ids for the file opened and assigned to file.variable 'BP'. Assigns the list to select.variable 'BLIST'.
READ A FROM FILEX,'ALIST' ELSE STOP SELECT A	Creates a select list of the attributes in item ALIST.

Sample usage of the SELECT statements.

The SEQ function converts an ASCII character to its corresponding numeric value.

FORMAT:

SEQ(expression)

The first character of the string value of the expression is converted to its corresponding numeric value. The following example will print the number 49:

```
PRINT SEQ('1')        (character 1 = ASCII 49 decimal)
```

Conversely, the CHAR function is available to convert a numeric expression to its corresponding ASCII character string value.

(See: CHAR)

NOTE: For a complete list of ASCII codes, refer to APPENDIX E.

STATEMENT	EXPLANATION
DIM C(50) S = 'THE GOOSE FLIES SOUTH' FOR I=1 TO LEN(STRING) C(I) = SEQ(S[I1]) NEXT I	Encodes in vector C elements the decimal equivalents of individual characters of character string S.

Sample Usage of the SEQ Function.

9.105 SIN FUNCTION : SINE OF AN ANGLE

The SIN function generates the trigonmetric sine of an angle.

FORMAT:

SIN(expression)

The SIN function generates the sine of an angle, expressed in degrees.

Values which are less than 0 degrees, or greater than 360 degrees are adjusted to this range before generation.

(See: COS)

STATEMENT	EXPLANATION
YY = SIN(XX)	Assigns the sine of an angle of XX degrees to YY.
PRINT SIN(1)	Prints "0.0174"
PRINT SIN(361)	Prints "0.0174"
PRINT SIN(2)	Prints "0.0349"
PRINT SIN(362)	Prints "0.0349"
PRINT SIN(45)	Prints "0.7071"
PRINT SIN(90)	Prints "1"

Sample usage of the SIN function.

9.106 SLEEP OR RQM STATEMENT : TIME ALLOCATION

The RQM or SLEEP statement terminates the executing program's current quantum (time-slice) The RQM or SLEEP statement may be used to effect program execution speed.

FORMAT:

RQM {seconds}	or	SLEEP {seconds}
RQM{"time.expression"}	or	SLEEP{"time.expression"}

The time-shared environment of the Pick system allows concurrent execution of several programs, with each program executing for a specific time period (called a time-slice or quantum) and then pausing while other programs continue execution. The RQM statement terminates the program's current time-slice. The RQM statement may be used in heavy compute loops to allow increased execution speed of other concurrently executing programs by giving up time. It may also be used to cause predetermined pauses (in seconds or until specified time) in program execution. The seconds parameter does not require quotes. The time expression (AM, PM or MILITARY) requires enclosure in quotes.

STATEMENTS	EXPLANATION
SLEEP 20	Sleep fo 20 seconds.
SLEEP "15:00"	Sleep until 3:00 PM.

```
  * PROGRAM SEGMENT TO SOUND
  * TERMINAL "BELL" FIVE TIMES.
  BELL=CHAR(7)
  FOR I=1 TO 5
  PRINT BELL:                    RQM statement allows enough
  RQM                            time for bell to be heard as
  NEXT I                         discrete "beeps".
  END
```

Sample usage of the SLEEP and RQM statements.

9.107 SPACE FUNCTION : STRING SPACING

The SPACE function generates a string value containing a specified number of blank spaces.

FORMAT:

SPACE(length)

the SPACE function generates a string value containing the number of blank spaces specified by the length. For example:

```
PRINT SPACE(10):"HELLO"
```

This statement prints 10 blanks followed by the string "HELLO".

Conversely, the TRIM function is available to delete extraneous blanks.

(See: TRIM)

STATEMENT	EXPLANATION
B = 14 A = SPACE(B)	Assigns to variable A the string value containing 14 blank spaces.
DIM M(10) MAT M = SPACE(20)	Assigns a string consisting of 20 blanks to each of the 10 elements of array M.
S = SPACE(5) L = "SMITH" C = "," F = "JOHN" N = S:L:S:C:S:F	Assigns to variable N the concatenated string consisting of 5 blanks, the name SMITH, 5 blanks, a comma, 5 blanks, and the name JOHN, or " SMITH , JOHN"

Sample Usage of the SPACE Function.

9.108 SQRT FUNCTION : SQUARE ROOT CABABILITY

The SQUARE ROOT function returns the positive square root of a positive number.

FORMAT:

SQRT(expression)

The SQUARE ROOT function returns the positive square root of any positive number (expression) that is greater than or equal to 0 and less than or equal 14,073,748,835 at precision 4.

STATEMENT	EXPLANATION
Y = SQRT(36)	Assign the value 6 to variable Y.
PRINT SQRT(1024)	Prints "32".
PRINT SQRT(1000)	Prints "31.6227"
PRINT SQRT(14073748834)	Prints "118632.832"

Sample Usage of the SQRT Function.

9.109 STOP STATEMENT : TERMINATION

The STOP statement may appear anywhere in the program; it designates a logical termination of the program.

FORMAT:

```
STOP {errnum{,param, param, ...}}
```

Upon the execution of a STOP statement, the PICK/BASIC program will terminate. If the program was called from a PROC the control will be returned to the calling PROC.

The STOP statement may be placed anywhere within the PICK/BASIC program to indicate the end of one of several alternative paths of logic.

The STOP statement may optionally be followed by an error message name, and error message parameters separated by commas. The error message name is a reference to an item in the ERRMSG file. The parameters are variables or literals to be used within the error message format.

(See: ABORT)

```
      A=500 ; B=750 ; C=235 ; D=1300
      REVENUE=A+B  ;  COST=C+D
      PROFIT=REVENUE-COST
      IF PROFIT > 0 THEN GOTO 10
      PRINT "ZERO PROFIT OR LOSS"
      STOP <-------------------------- If this path taken,
 10   PRINT "POSITIVE PROFIT"          Program will terminate
      END                              because profit is less
                                       than 0.
```

Sample usage of the STOP Statement.

9.110 STR FUNCTION : GENERATING STRING VALUES

The STR function generates a string value containing a specified number of occurrences of a specified string.

FORMAT:

STR(expression,occurence#)

The STR function generates a string value containing the number of occurrences specified by the occurence# of the string specified by the expression. The following statement, for example, assigns a string value containing 12 asterisk characters to variable X:

```
X=STR('*',12)
```

As a further example, the following statement will cause the string value "ABCABCABC" to be printed:

```
PRINT STR('ABC',3)
```

STATEMENT	EXPLANATION
VAR = STR("A",5)	Assigns to variable VAR the string value containing five A's.
A = 'BBB' B = STR("B",3) C = B CAT A	Assigns to variable C the string value containing six B's.
N = STR("?%?",4)	Assigns to variable N the string value containing 4 consecutive occurrences of the string "?%?".

Sample Usage of the STR Function.

9.111 SYSTEM FUNCTION : CALLING PRE-DEFINED SYSTEM VALUES

The SYSTEM function allows the user to obtain certain pre-defined values from the system. The value returned may either be an error status code (generated as a result of a previous BASIC statement), or a parameter such as the page-number or page-width.

FORMAT:

SYSTEM(expression)

The value of expression is in the range 0 through the maximum value as defined in table A. If the value of "expression" is outside the allowable range, the SYSTEM function will return a value as if the "expression" evaluated to zero (the error function).

If the expression used in the SYSTEM function is a zero, the function returns a value determined by the last executed BASIC statement that set an error condition. Examples of such BASIC statements are the tape commands such as READT, WRITET, etc. where the ELSE branch executes. SYSTEM(0), therefore, allows one to determine exactly what error has occurred when the program follows the ELSE branch of these statements. If the ELSE branch was not followed, the value returned by SYSTEM(0) is zero.

For example, the sequence of BASIC instructions:

```
READT TAPERECORD ELSE
    BEGIN CASE
      CASE SYSTEM(0) = 1; PRINT "ATTACH THE TAPE UNIT"; STOP
      CASE SYSTEM(0) = 2; PRINT "END OF FILE; DONE!"; STOP
    END CASE
END
```

will result in one of the messages being printed if the tape unit was not attached to the line running the BASIC program or if an EOF is read from the tape.

The SYSTEM function, with non-zero values of the expression, returns parameters that have been set external to the BASIC program. See Table A.

SYSTEM expression	Value returned
0	Error function value; see table B.
1	1 If PRINTER ON or (P) option used in RUN; 0 If data is being printed to the terminal.
2	Current page-size from TERM statement (page-width in columns).
3	Current page-depth from TERM statement (number of lines in page).
4	Number of lines remaining in current page.
5	Current page-number.
6	Current line-counter (number of lines printed).
7	One-character terminal-type code from TERM statement.
8	Current tape record length.
9	Current CPU millisecond count.
10	1 if current stack (STON) condition enabled. 0 if current stack inactive.
11	1 if a SELECT-LIST is active. 0 if a SELECT-LIST function is inactive.
12	Current time in milliseconds
13	Forces an RQM and returns 1.
14	Number of bytes in terminal input buffer.
15	Returns verb options as a character string.
16	Returns nested EXECUTE level
17	Returns error message string, with each number separated by an attribute mark.
100	Returns current release, version, and version date.

Meaning of values usable in the SYSTEM function.

Previously executed BASIC statement.	Error code returned.	Meaning
READT, WRITET, WEOF or REWIND	1	Tape unit is not attached.
READT	2	EOF read from tape unit.
WRITET	3	Attempted to write null string.
WRITET	11	Attempted to write variable longer than tape record length.

Values returned by the error function, SYSTEM(0)

9.112 TAN FUNCTION : TANGENT OF AN ANGLE

The TAN function generates the trigonmetric tangent of an angle.

FORMAT:

TAN(expression)

The TAN function generates the tangent of an angle, expressed in degrees.

Values which are less than 0 degrees, or greater than 360 degrees are adjusted to this range before generation.

(See: COS and SIN)

STATEMENT	EXPLANATION
YY = TAN(XX)	Assigns the tangent of an angle of XX degrees to yYY.
PRINT TAN(1)	Prints "0.0174"
PRINT TAN(361)	Prints "0.0174"
PRINT TAN(2)	Prints "0.0349"
PRINT TAN(362)	Prints "0.0349"
PRINT TAN(45)	Prints "1"
PRINT TAN(90)	Prints "0"

Sample usage of the TAN function.

9.113 TIME() AND TIMEDATE() FUNCTIONS : TIME AND DATE

The TIME() function returns the internal time of day. The TIMEDATE() function returns the current time and date in external format.

FORMAT:

```
TIME()

TIMEDATE()
```

The TIME() function returns the string value containing the internal time of day. The internal time is the number of seconds past midnight.

For example, at 4 minutes and 18 seconds after 5 P.M., the following statement would print ' 61458 '. (17:04:18 is 61458 seconds since midnight.)

```
PRINT TIME()
```

The TIMEDATE() function returns the string value containing the current time and date in the external format. This format is:

```
HH:MM:SS  DD MMM YYYY
          or
17:04:18  01 APR 1985
```

(See: DATE() function)

STATEMENT	EXPLANATION
A = TIME()	Assigns string value of current internal time to variable A.
IF TIME() > 1000 THEN GOTO 10	Branches to statement 10 if more than 1000 seconds have passed since midnight.
PRINT TIMEDATE()	Prints the current time and date in the external format.
WRITET TIME() ELSE STOP	Writes the string value of the current internal time onto a magnetic tape record.

Sample usage of the TIME() and TIMEDATE() functions.

9.114 TRIM FUNCTION : DELETING EXTRANEOUS SPACES

The TRIM function removes extraneous blank spaces from a specified string.

FORMAT:

TRIM(expression)

The TRIM function deletes preceding, trailing, and redundant blanks from the literal or variable expression. For example:

```
A='       GOOD MORNING,       MR. BRIGGS        .
A=TRIM(A)
PRINT A
```

The PRINT statement will print:

GOOD MORNING, MR. BRIGGS

Conversely, the SPACE function is available to generate blank spaces.

(See: SPACE)

STATEMENT	EXPLANATION
S = SPACE(5) L = "SMITH" C = "," F = "JOHN" JOHN'. N = S:L:S:C:S:F	Assigns to variable N the concatenated string consisting of 5 blanks, the name SMITH, 5 blanks, a comma, 5 blanks, and the name JOHN, or 'SMITH ,
M = TRIM(N)	Assigns to variable M a string consisting of the name SMITH, 1 blank, a comma, one blank, and the name JOHN, or 'SMITH , JOHN'

Sample Usage of the TRIM Function.

9.115 UNLOCK STATEMENT : CLEARING EXECUTION LOCKS

The UNLOCK statement provides a file and execution lock clearing capability for PICK/BASIC programs. The LOCK statement sets execution locks while the UNLOCK statement releases them.

FORMAT:

UNLOCK (expression)

The LOCK statement sets an execution lock so that when any other BASIC program attempts to set the same lock, then that program will either execute an alternate set of statements or will pause until the lock is released via an UNLOCK statement by the program which originally locked it.

The value of the expression specifies which execution lock is to be released (cleared). If the expression is omitted, then all execution locks which were previously set by the program will be released.

All execution locks set by a program will automatically be released upon termination of the program.

(See: LOCK)

STATEMENTS	EXPLANATION
UNLOCK 47	Resets execution lock 47.
UNLOCK	Resets all execution locks previously set by the program.
UNLOCK (5+A)*(B-2)	The value of the expression specifies which execution lock is released.

Sample Usage of the UNLOCK Statement.

9.116 WEOF STATEMENT : POSITIONING TAPE

BASIC programs may specify Magnetic Tape positioning operations through the use of the WEOF (Write End-of-File Mark) statement.

FORMAT:

WEOF THEN/ELSE statements

The WEOF statement writes two EOF (END OF FILE) marks on the tape, then backspaces over the second one. This correctly positions the tape for subsequent WRITET operations.

(See: WRITET)

STATEMENT	EXPLANATION
WEOF ELSE STOP	Writes two EOF marks, then backspaces over the second one.

Sample usage of the WEOF statement.

9.117 WRITE STATEMENT : MODIFYING ITEMS

The WRITE statement is used to update a file item.

FORMAT:

WRITE expression ON {file.variable,} itemname

The WRITE statement replaces the content of the item specified by itemname with the string value of the expression. The optional file.variable specifies the file variable; if it is used, the item will be replaced in the file previously assigned to that file.variable via an OPEN statement. If the file.variable is omitted, then the internal default variable is used (thus specifying the file most recently opened without a file variable). If the itemname specifies an item which does not exist, then a new item will be created.

The user should note that the PICK/BASIC program will abort with an appropriate error message if the specified file has not been opened prior to the execution of the WRITE statement.

(See: WRITEV and WRITET)

STATEMENT	EXPLANATION
WRITE "XXX" ON A, "ITEM5"	Replaces the current content of item ITEM5 (in the file opened and assigned to variable A) with string value "XXX".
A="123456789" B="X55" WRITE A ON FN1,B	Replaces the current content of item X55 (in the file opened and assigned to variable FN1) with string value "123456789".
WRITE 100*5 ON "EXP"	Replaces the current content of item EXP (in the file opened without a file variable) with string value "500".

Sample Usage of the WRITE Statement.

9.118 WRITET STATEMENT : WRITING RECORDS TO TAPE

BASIC programs may specify Magnetic Tape output operations through the use of the WRITET (Write Tape Record) statement. The record length on the tape is as specified by the most recent T-ATT statement executed at the TCL level.

FORMAT:

WRITET expression THEN/ELSE statements

The WRITET statement writes a record onto the magnetic tape. The string value of the expression is written onto the next record of the tape.

If the tape unit has not been attached, or if the string value of the expression is the empty string (''), then the statement(s) following the ELSE will be executed.

(See: T-ATT,READT, and system)

STATEMENT	EXPLANATION
FOR I=1 TO 5 WRITET A(I) ELSE STOP NEXT I	The values of array elements A(1) through A(5) are written onto 5 tape records. If one of the array elements has a value of '' (or if tape unit not attached), the program will terminate.

Sample Usage of the WRITET statement.

The WRITEU and WRITEVU statements have the letter "U" appended to them to imply update. The execution of these commands will not unlock the group locked by the program.

FORMAT:

```
WRITEU variable ON {file.variable,} itemname

WRITEVU variable ON {file.variable,} itemname,att#
```

These statements execute identically to the WRITE and WRITEV statements, with the following noted additional functionality.

(See: WRITE and WRITEV)

This version of these commands will not unlock the group locked by the program. This varient is used primarily for master file updates when several transactions are being processed and an update of the master item is made following each transaction update.

If the group is not locked when the WRITEU, WRITEVU or MATWRITEU statement is executed, the group will not be locked by the execution of the command.

STATEMENT	EXPLANATION
WRITEU CUST.NAME ON CUST.FILE,ID	Replaces the current contents of the item specified by variable ID (in the file opened and assigned to variable CUST.FILE) with with the contents of CUST.NAME. Does not unlock the group.
WRITEVU CUST.NAME ON CUST.FILE,ID,3	Replaces the third attribute of item ID (in the file opened and assigned to variable CUST.FILE) with the contents of variable CUST.NAME. Does not unlock the group.

Sample usage of WRITEU and WRITEVU statements.

9.120 WRITEV STATEMENT : UPDATING AN ATTRIBUTE

The WRITEV statement is used to write (update) a single attribute value to an item in a file.

FORMAT:

WRITEV expression ON {file.variable,} itemname,att#

Upon the execution of the WRITEV statement, the value of the expression becomes the attribute specified by att# (attribute number), in the item specified by the itemname and in the file previously assigned to the specified file.variable via an OPEN statement.

If the file.variable is omitted, then the internal default variable will be used (thus specifying the file most recently opened without a file.variable).

If a non-existent item name (or attribute number) is specified, then a new item (or attribute) will be created.

The WRITEV statement will also allow the attribute number (att#) to have a value of either zero or minus one, thus inserting data prior to the first attribute or following the last attribute.

When att# = 0, the expression is inserted at the begining of the item. All attributes in the item are shifted by 1 attribute and the expression becomes attribute 1.

When att# = -1, the expression is appended to the end of the item. The number of attributes in the item increase by 1 and all previously existing attributes are undisturbed.

The PICK/BASIC program will abort with an appropiate error message if the specified file has not been opened prior to the execution of the WRITEV Statement.
(See: WRITE and MATWRITE)

STATEMENT	EXPLANATION
Y="THIS IS A TEST" WRITEV Y ON X,"PROG",0	The string value "THIS IS A TEST" is inserted prior to the first attribute of item PROG in the file opened and assigned to variable X.
WRITEV "XYZ" ON "A7",4	Attribute 4 of item A7 (in the file opened without a file variable) is replaced by string value "XYZ".

Sample usage of the WRITEV statements.

9.121 XTD FUNCTION : HEXADECIMAL TO DECIMAL CONVERSION

The XTD function converts a value from Hexadecimal to Decimal.

FORMAT:

XTD(expression)

The string value of the expression is converted from Hexadecimal to Decimal. For example:

```
B = XTD(A)
```

Conversely, the DTX function is available to convert string values from Decimal to Hexadecimal.

(See: DTX)

STATEMENT	EXPLANATION
D = XTD(H)	Assigns the Decimal value of variable H to variable D.

Sample Usage of the XTD function.

9.122 PICK/BASIC SYMBOLIC DEBUGGER : AN OVERVIEW

The PICK/BASIC Symbolic Debugger facilitates the debugging of new PICK/BASIC programs and the maintenance of existing PICK/BASIC programs.

When a PICK/BASIC program is compiled a symbol table item is automatically generated unless the suppress option (S) has been used. This table is used by the PICK/BASIC Debugger to reference symbolic variables during program execution.

The PICK/BASIC Debugger may be entered at execution time by 1) depressing the BREAK key or 2) using the 'D' (debug) option with the RUN verb. Once the PICK/BASIC Debugger has entered it will indicate the source code line number about to be executed and will prompt for commands with an asterisk (*) as opposed to the System Debugger prompt '!' or the TCL prompt.

The user now has at his disposal the following general capabilities: Controlled stepping through execution of program by way of single or multiple steps. Transferring control to a specified step (line number). Breaking (temporary halting) of execution on specified line number(s) or on the satisfaction of specified logical conditions. Displaying and/or changing any variable(s), including dimensioned variables. Tracing variables. Conditional entry to the System Debugger. Directing output (terminal/printer). Stack manipulation (displaying and/or popping the stack). Displaying of specified (or all) source code line(s).

The symbol table is embedded in the object code which is placed in the catalog space. The debugger has instant access to the symbol table, and requires the use of the 'Z' command only when access to the source code is required. Note that the user may suppress generation of the symbol table by using the (S) option when compiling programs.

A user requires SYS2 privileges to use the PICK/BASIC debugger. This prevents users from making unauthorized changes to data during reporting and data entry.

BASIC DEBUGGER FEATURE	RELATED COMMAND
1. Set breakpoint on logical condition where 'o' is logical operator <,>,=,#, 'v' is variable, 'c' is condition to be met, or 'n' is line number where preceded by B_$o.	Bvoc{_.ul oc} or B$on
2. Display breakpoint table	D
3. Escape to System Debugger	DEBUG or DE
4. Single/multiple step execution	E{n}
5. End program execution and return to TCL	END
6. Proceed from breakpoint to specified line 'n'	G Gn
7. Remove all breakpoints specified breakpoint 'n'	K Kn
8. Display source code current line 'n' number of lines from current one number of lines from m-n all lines	L Ln Lm-n L*
9. Toggle output device (terminal & printer)	LP
10. Pass one breakpoint before stopping 'n' breakpoints	N Nn
11. Logoff	OFF
12. Inhibit output	P
13. Printer-close output to spooler	PC
14. Pop return stack Display return stack	R S
15. Switch turns trace table on/off Trace specified variable 'v'	T Tv
16. Remove all traces specified trace	U Un
17. Request symbol table	Z
18. Display current line number Print value of variable 'v' of element 'x' in array 'm' of element 'x,y' in matrix 'm' of entire array 'm' entire symbol table	$ /v /m(x) /m(x,y) /m /*

9.122.1 USING THE PICK/BASIC DEBUGGER : AN EXAMPLE

The following is a step-by-step introduction to the use of the PICK/BASIC DEBUGGER for inexperienced users. This will demonstrate only a few of the commands as it is merely intended to give the user an introductory "feeling" for the use of the PICK/BASIC DEBUGGER.

A sample program "SAMPLE" is shown below, followed by steps a user might take to debug it.

SAMPLE

```
001 DIM ARRAY(10) ; * ARRAY HAS 10 SLOTS
002 FOR I = 1 TO 20 ; * BUG: LOOP SPECIFIES 20 PASSES, ARRAY HAS ONLY 10
003   ARRAY(I) = I ; * EACH SLOT WILL BE FILLED WITH A CONSECUTIVE #
004 NEXT I
005 PRINT ARRAY(I)
006 END
```

"SAMPLE" compiles without any errors detected. Once it is run however, it aborts with the error message "ARRAY SUBSCRIPT OUT OF RANGE" and traps to the PICK/BASIC DEBUGGER. Supposing that the user cannot find the error, the following steps could be taken for detecting the error using the PICK/BASIC DEBUGGER.

1. The user enters the command "Z" to the DEBUGGER prompt character "*". The DEBUGGER responds with "PROG NAME?", the user enters the program name. This allows the DEBUGGER access to the symbol table created during compilation. Alternatively, if the user uses the debug option "(D)" during run time, access to the symbol table is already established, and use of the "Z" command is unnecessary.

2. To find out how far in the loop the program progressed, the user looks at the variable "I" by entering "/I". The DEBUGGER responds with "11 =", at which the user may change the value of "I" if desired. The user may then want to look at all of the values in the array by entering "/ARRAY". The DEBUGGER responds with "ARRAY(1)=1=", the user depresses return and the DEBUGGER continues with the next "array slot" (i.e., "ARRAY(2)=2=" etc.). Once "ARRAY(10)=10=" has been reached the user presses return and the DEBUGGER returns with the "*" prompt, the user knows that the array has only 10 slots and the loop calls for 20 -- thus he finds the error. The user may then end the "session" with PICK/BASIC DEBUGGER by entering "END" and repair the bug.

A summary of this interaction is given in Figure A on the next page. For purposes of clarity, whatever is entered by the user is shown enclosed in square brackets "[]". These are <u>not</u> part of the commands; they are to distinguish user entry from DEBUGGER response.

NOTE: The square brackets surround user input to distinguish it from PICK/BASIC Debugger responses. They are not part of the commands!

```
LINE 3 (B17) ARRAY SUBSCRIPT OUT OF RANGE
*I3
*[/I]  <RETURN> 11-
*[/ARRAY]  <RETURN> ARRAY(1)-1-  <RETURN>
  ARRAY(2)-2-  <RETURN>
  ARRAY(3)-3-  <RETURN>
  ARRAY(4)-4-  <RETURN>
  ARRAY(5)-5-  <RETURN>
       .
       .
       .
  ARRAY(10)-10-  <RETURN>
*<RETURN>
*[END]
```

Sample Session with the PICK/BASIC Debugger.

NOTE: A carriage return will return control to the BASIC DEBUGGER prompted by "*" whereas a line-feed will return control to program execution until a breakpoint, an error or the end of the program is met.

9.122.2 THE TRACE TABLE

The trace table is used for the automatic printout of a specified variable or variables after a break has occured.

Up to six trace values may be entered in the table. Either the symbolic name, or a line number and variable number may be used to reference the variable. In addition, all the variables in the last statement executed may be printed out. The trace table may be alternately turned on and off by use of the "T" return command.

Examples of use of the trace table are shown below:

Tname	The value of the variable name will be printed out at each breakpoint.
T%10,3	The value of the third variable in line number 10 will be printed out at each breakpoint. If line number 10 contains the statement "A=B+C+D" the value of "C" will be printed.

To delete a variable from the trace table use the "U" command followed by the trace variable to be deleted. For example, to delete the variable name from the table type in "Uname". "U" return deletes the entire trace table.

If a program calls an external subroutine, and the BASIC/DEBUGGER has been entered previously, a complete symbol table will be set up for the external subroutine. the table will have 4 break-points and 6 variable traces available, as well as pointers to program source and object, which may be set up by the Z command. break points set up for a subroutine are independent from break points set up in the main program or other subroutines; however, the execution counters (E and N,) are global.

The use of multiple symbol tables allows the programmer to set up different break points and/or variable traces for different subroutines.

Commands to set (B)reakpoints, (D)isplay and (K)ill breakpoints.

Command:

B[variable-name][operator][expression]{ & another condition}
B$[operator][line-number]{ another condition}

Where 'variable-name' is a simple variable or an explicitly stated array element and 'expression' is a variable, constant, or array element. If the variable does not exist or if the wrong Symbol Table is assigned, the message "SYM NOT FND" will be printed. String constants must be enclosed in quotes using the same rules that apply to PICK/BASIC literals. The Breakpoint Table may contain up to four conditions that when satisfied, will cause a break in execution. Logical expressions are used to set the break conditions. The logical operators used are:

<	less than
>	greater than
=	equal to
#	not equal to
&	is used as a logical connector between conditions.
$	is a special symbol used to indicate that a line number is specified rather than a variable name.

A plus sign will be printed next to the command if it is accepted. When the condition is met, an execution break will occur and the Debugger will halt execution of the program and print *Bn 1 where 'n' is one of the 4 Breakpoint Table entries and '1' is the program line number that caused the break.

Command:	D	Displays Trace and Breakpoint Tables
Command:	Kn	Deletes nth breakpoint, n in range 1 to 4
	K<RETURN>	Deletes all breakpoint conditions

The 'K' command is used to delete breakpoint conditions from the table. A minus sign will be printed next to the command to indicate that an entry has been removed.

COMMAND	EXPLANATION
BX<42	Breaks when X is less than 42.
BADDRESS=''	Breaks when ADDRESS is null.
BDATE=INV.DATE&$=22	Breaks when variable DATE is equal to variable INV.DATE and if the line number is 22.
K2	Kills the second breakpoint condition.
BPRICE(3)=24.98	Breaks when the third element of the array PRICE is equal to 24.98. Only individual array elements may be specified.
D	Displays the Trace and Breakpoint Tables.
K	Kills all breakpoint conditions.

Examples of B, D, and K Commands.

The commands "E", "G", and "N" in conjunction with the breakpoint table control the execution of the program under debug control.

The "E" command will allow the execution of a specified number of lines before returning control to the user. "E" return will turn off the "E" command.

Command: E Execution continues until interuption by the user, by a breakpoint or until program ends.

ES Program will enter the debugger after executing five program lines.

The "N" command will allow the user to bypass any number of breakpoints before control is passed back to the user, however, the trace table variables will be printed at each breakpoint. "N0" equals 'pass one breakpoint', "N1" equals 'pass two breakpoints', etc. and "N" return will set "N" to "N0".

Command: N3 Four breakpoints are passed, although the trace table values, if present, are printed out at each breakpoint. Control is then returned to the user.

The "G" command followed by a line number will allow control to be passed to the line number indicated. The "G" return command will cause program to execute the next command from the current line number and it will return control depending on the breakpoint setup.

Command: G153 Control passes to line number 153 and thereafter to user.

G Control passes to next program line and thereafter to user.

9.122.5 SLASH '/' COMMAND : DISPLAYING AND CHANGING VARIABLES

Variables and arrays can be displayed and changed in either decimal or string formats.

To display a variable use the command '/v' where 'v' is a variable. For example to display the value of the variable name select '/name'. The DEBUGGER will respond with the string in the name field and an equal sign. If the variable is not to be changed press return. If the variable is to be changed put in the new value of the variable desired and press return. To display a complete array just place the name of the array after the slash. To display one value in the array use the form '/M(x)' or '/M(x,y)' where 'x' and 'y' are points in the array. The array point may then be changed in the same way as for a single variable.

A window may be placed after any variable selection by following the variable with a ';' and the length of the window. For example, to limit the variable name to eight characters the command '/name;8' would be used. Numeric variables will ignore any window commands.

The symbolic name of the variable may be replaced with the form '%x,y' where 'x' is the line number and 'y' is the nth variable in that line in the same way as the breakpoint table. Examples of displaying and changing variables follows:

/CITY IRVINE=	The variable 'city' is displayed but not changed.
/STATE NY=CA	The variable 'state' is displayed as 'NY' and changed to 'CA'
/FIELD(5) 10=	The fifth point in array FIELD is displayed as 10 and not changed.
/*	All the symbols in the symbol table are displayed.

Additional PICK/BASIC Debugger commands : PC, P, LP, $, L, Z, [], DEBUG and END.

I/O CONTROL

The "P" command inhibits all PICK/BASIC program output so that the user may look only at the DEBUGGER output. "P" return alternately turns "P" on and off.

The "LP" command forces all output to the line printer which can be used for a fast trace or hard copy of a trace. "LP" return alternately turns the line printer command on and off.

The "PC" command is the same as the PICK/BASIC printer close command. All data that is waiting to be sent to the printer is output at this time.

SOURCE CODE DISPLAY

The "$" command will print the next line number to be executed.

The "L" command will display sorce code lines. "L" will display the current line of source. "Ln" will display line 'n'. "Lm-n" will display lines 'm-n'. "L*" will display the entire source program.

SYMBOL TABLE

The "Z" command will enable a symbol table other than the currently running programs' symbol table, which is the default. The program name may be entered as {file-name,{dataname}} item-name.

STRING WINDOWS

The string window command "[n,m]" will cause the output of all variables to be limited to the substring selected. An example of the command follows:

X=1234567890.B [3,2] Sets the window for the third character position with a string length of two. Any printout of x will be 34.

Setting the window length to zero will turn the string window command off. "[Carriage-return" will have the same result.

ESCAPE TO SYSTEM DEBUGGER

The "DEBUG" command will pass control to the System DEBUGGER.

TERMINATION

The "END" command will terminate the PICK/BASIC and DEBUG programs and return control to TCL.

9.122.7 GENERAL CODING TECHNIQUES : HELPFUL HINTS

This topic presents a number of general coding techniques which the programmer should keep in mind when writing PICK/BASIC programs.

The PICK system uses standard attribute and value delimiters. These should be defined once in the initialization portion of the program, and then referenced by their variable name. for example:

```
EQU AM TO  CHAR(254)Attribute Mark
EQU AM TO  CHAR(253)Value Mark
EQU SVM TO CHAR(252)Secondary Value Mark
```

Cursor positioning can be controlled by setting variables names to @ functions and then PRINTing those names in the body of your program.

```
ERASE.SCREEN = @(-1)                    HOME = @(-2)
CLEAR.TO.END.OF.SCREEN = @(-3)          CLEAR.TO.END.OF.LINE = @(-4)
START.BLINK = @(-5)                     STOP.BLINK = @(-6)
START.PROTECT = @(-7)                   STOP.PROTECT = @(-8)
BACKSPACE = @(-9)                       UP.1.LINE = @(-10)

DOWN.1.LINE = CHAR(10)                  RIGHT.1.CHARACTER = CHAR(6)
BELL = CHAR(7)
```

To erase the screen for instance, the PRINT statement would be:

```
PRINT ERASE.SCREEN
```

The OPEN statement is very time consuming and should be executed as few times as possible. All files should be opened to file variables at the beginning of the program; access to the files can then be performed by referencing the file variables.

The size of programs can be reduced, with a corresponding increase in overall system performance, by reducing the amount of literal storage. For example:

```
200 PRINT 'RESULT IS ':A+B
210 PRINT 'RESULT IS ':A-B
220 PRINT 'RESULT IS ':A*B
230 PRINT 'RESULT IS ':A/B
```

These statements should have been written as follows:

```
        MSG = 'RESULT IS'
        .
        .
200 PRINT MSG:A+B
210 PRINT MSG:A-B
220 PRINT MSG:A*B
230 PRINT MSG:A/B
```

Operations should be pre-defined rather than repetitively performed. This operation, for example:

```
X=SPACE(9-LEN(OCONV(COST,'MCA'))):OCONV(COST,'MCA')
```

should have been written as follows:

```
E=OCONV(COST,'MCA')
X=SPACE(9-LEN(E)):E
```

In the same context, the following operation:

```
FOR I=1 TO X*Y+Z(20)
  .
  .
  .
NEXT I
```

should have been written as follows:

```
TEMP=X*Y+Z(20)
FOR I=1 TO TEMP
  .
  .
  .
NEXT I
```

The following LOOP construct could be used to access an unknown number of multivalues from an attribute (including null values):

```
EQU VM TO CHAR(253)
READV ATTR FROM ID, ATTNO ELSE STOP
VNO=1
LOOP
   VALUE=FIELD(ATTR,VM,VNO)
WHILE COL2() #0 DO
   PRINT VALUE
   VNO=VNO+1
REPEAT
```

9.122.8 PROGRAMMING EXAMPLES: PYTHAG

```
************************************************************
*  THIS PROGRAM FINDS PYTHAGOREAN TRIPLES
************************************************************
      PRINT
      PRINT 'SOME PYTHAGOREAN TRIPLES ARE:'
      PRINT
      FOR A=1 TO 40
         D=A-1
         FOR B=1 TO A-1
            CC=(A^2)+(B^2)N
            IF C = INT(C) THEN PRINT B,A,C  C=SQRT(EC)
         NEXT B
      NEXT A
      STOP
```

9.122.9 PROGRAMMING EXAMPLES: GUESS

```
*******************************************************
*  THIS PROGRAM IS A GUESSING GAME
*******************************************************
      HEADING ''
      HISSCORE=0; YOURSCORE=0
10    PAGE
      PRINT 'GUESS NUMBERS BETWEEN 0 AND 100'
      PRINT 'MACHINE:':HISSCORE:'    ':'YOU:':YOURSCORE
      PRINT
      NUM=RND(101)
      FOR I=1 TO 6
         PRINT 'GUESS ':I:' ':
         INPUT GUESS
         IF GUESS=NUM THEN
            PRINT
            PRINT 'CONGRATULATIONS, YOU WON!'
            YOURSCORE=YOURSCORE+1
            GOTO 60
         END
         IF GUESS < NUM THEN PRINT 'HIGHER'
         IF GUESS > NUM THEN PRINT 'LOWER'
      NEXT I
      PRINT
      PRINT 'YOU LOST YOU DUMMY, YOUR NUMBER WAS ':NUM
      HISSCORE=HISSCORE+1
60    PRINT
      PRINT 'AGAIN?':
      INPUT X
      IF X = 'NO' THEN STOP
      GOTO 10
   END
```

9.122.10 PROGRAMMING EXAMPLES: INV-INQ

```
*****************************************************************
*  THIS PROGRAM QUERIES AN INVENTORY FILE.
*  IT READS THE DICTIONARY OF FILE 'INV' TO GET THE ATTRIBUTE
*  NUMBERS OF 'DESC' (DESCRIPTION) AND 'QOH' (QUANTITY-ON-HAND).
*  THE PROGRAM THEN PROMPTS THE USER FOR A PART-NUMBER WHICH
*  IS THE ITEM-ID OF AN ITEM IN 'INV' AND USES THE ATTRIBUTE
*  NUMBERS TO READ AND DISPLAY THE PART DESCRIPTION AND
*  QUANTITY ON HAND. THE PROGRAM LOOPS UNTIL A NULL PART
*  NUMBER IS ENTERED.
*****************************************************************
*
*--- GET ATTRIBUTE DEFINITIONS FROM DICTIONARY OF INVENTORY FILE
     OPEN 'DICT','INV' ELSE PRINT 'CANNOT OPEN "DICT INV"'; STOP
     READV DESC.AMC FROM 'DESC',2 ELSE PRINT 'CANT READ "DESC" ATTR'; STOP
     READV QOH.AMC  FROM 'QOH',2  ELSE PRINT 'CANT READ "QOH"  ATTR'; STOP
*--- OPEN DATA PORTION OF INVENTORY FILE
     OPEN '','INV' ELSE PRINT 'CANNOT OPEN "INV"';  STOP
*--- PROMPT FOR PART NUMBER
100  PRINT
     PRINT 'PART-NUMBER ':
     INPUT PN
     IF PN = '' THEN PRINT '--DONE--';  STOP
     READV DESC FROM PN,DESC.AMC ELSE PRINT 'CANT FIND THAT PART';GOTO 100
     READV QOH  FROM PN,QOH.AMC  ELSE QOH=0
*--- PRINT DESCRIPTION AND QUANTITY-ON-HAND
     PRINT 'DESCRIPTION - ': DESC
     PRINT 'QTY-ON-HAND - ': QOH
     PRINT
     GOTO 100
  END
```

```
*****************************************************************
*  THIS PROGRAM FORMATS A PICK/BASIC PROGRAM TO
*  DISPLAY BLOCK STRUCTURING BY INDENTING LINES.
*****************************************************************
*---- DEFINITIONS
10    SP = 6           ;* LEFT MARGIN COLUMN NUMBER
      ID = 3           ;* NUMBER OF SPACES TO INDENT
*---- INITIALIZATION
      SPX = SP
      LINE.NO = 0
*---- INPUT FILE NAME AND  PROGRAM NAME
      PRINT
      PRINT
      PRINT 'DATA/BASIC FILE NAME    - ':; INPUT FILE
      IF FILE = '' THEN STOP
      OPEN '',FILE ELSE PRINT 'CANNOT OPEN FILE - ': FILE;  GOTO 10
      PRINT 'DATA/BASIC PROGRAM NAME - ':; INPUT NAME
      IF NAME = '' THEN GOTO 10
      NEWITEM = ''
      READ ITEM FROM NAME ELSE
         PRINT 'CANNOT FIND THAT PROGRAM'
         GOTO 10
      END
*---- GET NEW LINE, IF NONE - THEN DONE
100   LINE.NO = LINE.NO + 1
      LINE = EXTRACT(ITEM,LINE.NO,0,0)
      IF LINE = '' THEN
         WRITE NEWITEM ON NAME
         PRINT; PRINT; PRINT '--DONE--'; GOTO 10
      END
      LABEL = ''
*---- STRIP OFF LEADING/TRAILING SPACES
200   IF LINE[1,1] = ' ' THEN LINE = LINE[2,32767]; GOTO 200
210   IF LINE[LEN(LINE),1] = ' ' THEN LINE = LINE[1,LEN(LINE)-1]; GOTO 210
*---- LOOK FOR A COMMENT ('*', '!', OR 'REM')
      IF LINE[1,1] = '*' THEN GOTO 1500
      IF LINE[1,1] = '!' THEN GOTO 1500
      IF LINE[1,3] = 'REM' THEN GOTO 1500
*---- LOOK FOR 'FOR'
      IF LINE[1,4]='FOR ' AND INDEX(LINE,'NEXT ',1)>0 THEN GOTO 2000
      IF LINE[1,4]='FOR ' AND INDEX(LINE,'NEXT ',1)=0 THEN GOTO 1000
*---- LOOK FOR 'END'
      IF LINE = 'END' THEN GOTO 1100
      IF LINE[1,4] = 'END ' THEN
         IF LINE[LEN(LINE)-4,5] = ' ELSE' THEN GOTO 1200
      END
*---- LOOK FOR 'NEXT'
      IF LINE[1,5] = 'NEXT ' THEN GOTO 1100
*---- EXTRACT LEADING NUMERIC LABEL
      IF LINE[1,1] MATCHES '1N' THEN
         L = 2
300      IF LINE[L,1] MATCHES '1N' THEN L=L+1; GOTO 300
         LABEL = LINE[1,L-1]
         LINE = LINE[L,32767]
```



```
         GOTO 200
      END
*---- LOOK FOR LINE ENDING IN ' ELSE' OR ' THEN'  ('IF' OR 'READ')
      X = LINE[LEN(LINE)-4,5]
      IF X = ' THEN' THEN GOTO 1000
      IF X = ' ELSE' THEN GOTO 1000
*---- THIS IS JUST ANOTHER LINE, THEREFORE NO CHANGE
      GOTO 2000
*---- INDENT ON SUBSEQUENT LINES
1000  SP = SP + ID
GOTO 2000
*---- OUTDENT ON THIS AND SUBSEQUENT LINES
1100  SP = SP - ID
*---- OUTDENT THIS LINE ONLY
1200  SPX = SPX - ID
      GOTO 2000
*---- PRINT WITH NO INDENTATION
1500  SPX = 0
*---- WRITE NEW LINE
2000  NEW.LINE = LABEL : STR(' ',SPX-LEN(LABEL)) : LINE
      PRINT NEW.LINE
      NEWITEM = REPLACE(NEWITEM,LINE.NO,0,0,NEW.LINE)
      SPX = SP
      GOTO 100
   END
```

```
*********************************************************
*  THIS PROGRAM UPDATES DATA ON LOTS IN A HOUSING TRACT.
*  ITEM-ID'S IN "LOT" FILE ARE TRACT.NAME*LOT.NUMBER
*********************************************************
100* INITIALIZATION
      PROMPT '='
      CLEAR
      DIM DESC(30),TYPE(30)
      OPEN 'DICT','LOT' ELSE
         PRINT "CAN'T OPEN DICT LOT"
         STOP
      END
*
200* GET DESCRIPTIONS, CONVERSIONS
      FOR I = 1 TO 30
         READ DICT.ITEM FROM I ELSE
            PRINT "DICTIONARY ITEM '":I:"' NOT FOUND"
            GOTO 250
         END
         D = EXTRACT(DIC.ITEM,3,0,0)          ;* S/NAME--DESCRIPTION
         IF D # '' THEN DESC(I) = D:STR('.',15-LEN(D))
         IF C[1,2] = 'MD' THEN
            TYPE(I) + 'NUM'
            GOTO 250
         END
         IF C[1,1] = '0' THEN TYPE(I) = 'DATE'
250*
      NEXT I
*
*
      OPEN '','LOT' ELSE
         PRINT "CAN'T OPEN LOT FILE."
         STOP
      END
*
300* GET THE TRACT NAME
      PRINT
      PRINT "TRACT NAME.....":
      INPUT TRACT
      IF TRACT = 'STOP' OR TRACT = 'END' THEN STOP
      IF TRACT = '' THEN GOTO 300
      READ INFO FROM TRACT ELSE
         PRINT "TRACT '":TRACT:"' OT ON FILE"
         GOTO 300
      END
*
400* GET A VALID LOT NUMBER
      PRINT
      PRINT "LOT NUMBER.....":
      INPUT NUMBER
      IF NUMBER = '' THEN GOTO 400
      IF NUMBER = 'END' OR NUMBER = 'STOP' THEN GOTO 300
      IF NUM(NUMBER) = 0 THEN
         PRINT "MUST BE A NUMBER"
```



```
         GOTO 400
      END
      NUMBER = TRACT:'*':NUMBER
      READ ITEM FROM NUMBER ELSE
         ITEM = ''
         PRINT "NEW LOT"
      END
*
450*
      NOT.SOLD = 0
      FOR I = 1 TO 30
            GOSUB 1000                  ;* UPDATES THE I'TH ATTRIBUTE
         IF I = 10 THEN
            IF EXTRACT(ITEM,10,0,0) = '' THEN
               NOT.SOLD = 1
               I = 19
            END
         END
*
         IF I = 21 THEN
            IF NOT.SOLD THEN GOTO 500
         END
      NEXT I
*
      VERITY DATA & STORE
      PRINT
      PRINT"         OK      ":
      INPUT OK
      IF OK = '' THEN
            WRITE ITEM ON NUMBER
            GOTO 400
      END
      IF OK = 'L' THEN
         PRINT
         FOR L = 1 TO 30
            ATT = EXTRACT(ITEM,I,0,0)
            IF ATT = '' THEN GOTO 550
            PRINT DESC(L):
            IF TYPE(L) = 'DATE' AND NUM(DATE) THEN ATT = OCONV(ATT,'DO')
            IF TYPE(L) = 'NUM' AND NUM(ATT) THEN ATT = 0.01 * ATT
            PRINT ATT 'R################'
550*
         NEXT L
         GOTO 500
      END
      GOTO 400
*
1000* UPDATE'S THE I'TH ATTRIBUTE OF "ITEM"
      IF DESC(I) = '' THEN RETURN          ;* NOT NEEDED OR NOT FOUND
      PRINT DESC(I):
      CURRENT = EXTRACT(ITEM,I,0,0)
*
      IF TYPE(I) = 'NUM' THEN
1100* NEED A NUMBER (AMOUNT)
         PRINT CURRENT*.01  'R###############':
         INPUT RESPONSE
```

```
         IF RESPONSE = '' THEN RETURN      ;* JUST LOOKING
         I F RESPONSE = '' THEN
            ITEM = REPLACE(ITEM,I,0,0,'')
            RETURN                          ;* DELETE THIS ATT.
         END
         IF NUM(RESPONSE) = 0 THEN
            PRINT "MUST BE A NUMBER"
            GOTO 1100
         END
*
         ITEM = REPLACE(ITEM,I,0,0,RESPONSE*100)
         RETURN
      END
*
      IF TYPE(I) = 'DATE' THEN
1200* NEED A DATE
         PRINT OCONV(CURRENT,'DO') 'R################':
         INPUT RESPONSE
         IF RESPONSE = '' THEN RETURN      ;* JUST LOOKING
         IF RESPONSE = 'T' THEN
            DATE = DATE()
            GOTO 1250
         END
         IF RESPONSE = '' THEN
            ITEM = REPLACE(ITEM,I,0,0,'')  ;* DELETE THIS ATT.
            RETURN
         END
         DATE = ICONV(RESPONSE,'D')
         IF DATE = '' THEN
            PRINT "USE DATE FORMAT 'MONTH/DAY/YEAR'"
            GOTO 1200
         END
1250*
         ITEM = REPLACE(ITEM,I,0,0,DATE)
         RETURN
      END
1300* NO NECESSARY FORMATS
      PRINT CURRENT 'R################':
      INPUT RESPONSE
      IF RESPONSE = '' THEN RETURN
      IF RESPONSE = '' THEN RESPONSE = ''
      ITEM = REPLACE(ITEM               I,0,0,RESPONSE)
      RETURN
END
```

9.123 SUMMARY OF PICK/BASIC STATEMENTS

This summary presents the general form for each of the PICK/BASIC Statements. The statements are listed in alphabetical order.

STATEMENTS

ABORT{errnum{,param,param,...}}

BREAK ON/OFF

CALL @name(argument list)

CALL name(argument list)

CASE --- BEGIN CASE

```
            CASE expression
             stmts
            CASE expression
             stmts
             .
             .
        END CASE
```

CHAIN "any TCL command"

CLEAR

CLEARFILE {file.variable}

COM{MON} variable {,variable}

CRTexpression

DATA expression{,expression...}

DELETE {file.variable,} itemname

DIM variable(dimensions) {,variable(dimensions)}

ECHO ON/OFF

ENTER"cataloged.program"

END

EXECUTEexpression {CAPTURING var1} {RETURNING var2}

EQU{ATE} variable TO equate-variable {, ...}

```
FOOTING "text 'options' (text 'options')"

FOR . ..NEXT---FOR variable = exp TO exp (STEP exp)(WHILE/UNTIL exp)

            NEXT variable

GOSUB statement-label

GO(TO) statement-label

HEADING "text 'options' (text 'options')"

IF expression THEN stmts (ELSE stmts)

INPUT variable (:)

INPUT @(column,row) : variable ('mask')

INPUTERR expression

INPUTTRAP 'xx' GOTO n,n,n . . .

INPUTTRAP 'xx' GOSUB n,n,n...

INPUTNULL x

LOCATE('String',Item(,Att#(,Val#));index#(;sequence))THEN/ELSE stmts

LOCK expression (THEN/ELSE stmts)

LOOP (stmts) WHILE/UNTIL expression DO (stmts) REPEAT

MAT variable

MAT array.variable = expression

MAT array.variable = MAT array.variable

MATREAD array.variable FROM (file.variable,) itemname THEN/ELSE stmts

MATREADU array.variable FROM   (file.variable,)  expression THEN/ELSE

stmts

MATWRITE array.variable ON (file.variable,) expression

MATWRITEU array.variable  ON (file.variable,) expression

NEXT variable

NULL

ON expression GOTO/GOSUB statement-label, statement-label

OPEN ("DICT",) "filename" (TO file.variable) THEN/ELSE stmts
```

PAGE (expression)

PRECISION n

PRINT (**ON** expression) print-list

PRINTER ON/OFF

PRINTER CLOSE

PROCREADvariable THEN/ELSE statements

PROCWRITEvariable

PROMPT expression

READ variable **FROM** (file.variable,) itemname **THEN/ELSE** stmts

READNEXT variable (,vmc)(**FROM** select.variable) **THEN/ELSE** stmts

READT variable **THEN/ELSE/** stmts

READU variable **FROM** (file.variable,) itemname **THEN/ELSE** stmts

READV variable **FROM** (file.variable,) itemname,att# **THEN/ELSE** stmts

READVU variable **FROM** (file.variable,) itemname,att# **THEN/ELSE** stmts

RELEASE ((file.variable,) expression)

REM or * or !

RETURN

RETURN TO statement-label

REWIND THEN/ELSE stmts

RQM (seconds or "time")

SELECT (file.variable)(**TO** select.variable)

SLEEP (seconds or "time")

STOP (errnum(param,param,...))

SUBROUTINE name (argument list)

UNLOCK (expression)

WEOF THEN/ELSE (expression)

WRITE expression **ON** (file.variable,) itemname

WRITEU variable **ON** (file.variable,) itemname

WRITET expression THEN/ELSE stmts

WRITEV expression ON {file.variable,} itemname,att#

WRITEVU expression ON {file.variable,} itemname,att#

9.124 BASIC INTRINSIC FUNCTION SUMMARY

This summary presents the general form for each of the PICK/BASIC Intrinsic Functions. The functions are listed in alphabetical order. page referenced.

FUNCTION

@(column{,row})

ABS(expression)

ALPHA (expression)

ASCII(expression)

CHAR(expression)

COL()

COL2()

COS (expression)

COUNT (string,substring)

DATE()

DCOUNT (string,substring)

DELETE(da.variable,att#{,value#{,sub-value#}})

EBCDIC(expression)

EXP(expression)

EXTRACT(da.variable,att#{,value#{,sub-value#}})

FIELD(expression,delimiter,occurence#)

ICONV(expression,conversion)

INDEX(string,sub-string,occurence#)

INSERT(da.variable,att#{,value#{,sub-value#,}}{;}new.expression)

INT(expression)

LEN(expression)

LN

MOD(numerator,denominator)

NOT(expression)

NUM(expression)

OCONV(expression,conversion)

PWR(expression,power)

REM(numerator,denominator)

REPLACE(da.variable,att#{,value#{,sub-value#,}}{;}new.expression)

RND(expression)

SEQ(expression)

SIN (expression)

SPACE(length)

SQRT (expression)

STR(expression,occurence#)

TAN (expression)

TIME()

TIMEDATE()

TRIM(expression)

9.125 BASIC COMPILER ERROR MESSAGES

This summary presents a list of the error messages which may occur as a result of compiling a PICK/BASIC program.

ERROR NO.	ERROR MESSAGE	CAUSE
B0	PROGRAM 'xx' COMPILED. n FRAMES USED.	PICK/BASIC program compiled with no compilation errors. This is not an error; it simply informs that compilation is completed.
B100	COMPILATION ABORTED; NO OBJECT CODE PRODUCED	Compilation errors present.
B101	MISSING "END", NEXT", "WHILE", "UNTIL", "REPEAT", OR "ELSE"; COMPILATION ABORTED, NO OBJECT CODE PRODUCED	Compilation error present.
B102	BAD STATEMENT	Unrecognizable statement.
B103	LABEL "C" IS MISSING	Label indicated by GOTO or GOSUB was not found.
B104	LABEL "C" IS DOUBLY DEFINED	More than one statement was found beginning with the same label.
B105	"C" HAS NOT BEEN DIMENSIONED	Subscripted variable was not dimensioned.
B106	"C" HAS BEEN DIMENSIONED AND USED WITHOUT SUBSCRIPTS	Dimensioned variable used without subscripts.
B107	"ELSE" CLAUSE MISSING	ELSE clause is missing.
B108	"NEXT" STATEMENT MISSING	NEXT statement is missing in FOR-NEXT loop.
B109	VARIABLE MISSING IN "NEXT" STATMENT	Iteration variable is missing in NEXT statement.
B110	"END" STATEMENT MISSING	END statement is missing in multi-line IF statement.
B111	"UNTIL" OR "WHILE" MISSING IN "LOOP"	UNTIL or WHILE clause is missing in a LOOP statement.

	STATEMENT	
B112	"REPEAT" MISSING IN "LOOP" STATEMENT	REPEAT is missing in a LOOP statement.
B113	COLUMN B TERMINATOR MISSING	Garbage following a legal statement or quote missing.
B114	MAXIMUM NUMBER OF VARIABLES EXCEEDED	Using the default descriptor size of 10, the maximum number of variables (including array elements) is 3274.
B115	LABEL 'C' IS USED BEFORE THE EQUATE STMT	The equate-variable is referenced before it has been defined.
B116	LABEL 'C' IS USED BEFORE THE COMMON STMT	A common variable has been referenced before it is put in common.
B117	LABEL 'C' IS MISSING A SUBSCRIPT LIST	An array is referenced without a subscript list.
B118	LABEL 'C' IS THE OBJECT OF AN EQUATE STMT AND IS MISSING	
B119	WARNING - PRECISION VALUE OUT OF RANGE - IGNORED!	
B120	WARNING - MULTIPLE PRECISION STATEMENTS - IGNORED!	
B121	LABEL 'C' IS A CONSTANT AND CAN NOT BE WRITTEN INTO	
B122	LABEL 'C' IS IMPROPER TYPE	

9.126 BASIC RUN-TIME ERROR MESSAGES

This summary presents a list of the error messages which may occur as a result of executing a PICK/BASIC program. Warning messages indicate that illegal conditions have been smoothed over (by making an appropriate assumption), and do not result in program termination. Fatal error messages result in program termination.

WARNING MESSAGES

Error No.	Error Message	Cause
B10	VARIABLE HAS NOT BEEN ASSIGNED A VALUE; ZERO USED!	An unassigned variable was referenced. (A value of 0 is assumed.)
B11	TAPE RECORD TRUNCATED TO TAPE RECORD LENGTH!	An attempt was made to write more onto a tape record than the tape record length. (The record is truncated to tape record length.)
B13	NULL CONVERSION CODE IS ILLEGAL; NO CONVERSION DONE!	A string variable that should have a value is actually null.
B16	NON-NUMERIC DATA WHEN NUERIC REQUIRED; ZERO USED!	A non-numeric string was encountered when a number was required. (A value of 0 is assumed.)
B19	ILLEGAL PATTERN	Illegal pattern used with MATCH or MATCHES operator.
B20	COL1 OR COL2 USED PRIOR TO EXECUTING A FIELD STMT; ZERO USED!	COL1 or COL2 function used before FIELD function used. (A value of 0 is assumed.)
B24	DIVIDE BY ZERO ILLEGAL; ZERO USED!	Division by zero attempted. (A value of 0 is assumed.)
B209	FILE IS UPDATE PROTECTED	
B210	FILE IS ACCESS PROTECTED	

FATAL ERROR MESSAGES

Error No.	Error Message	Cause
B12	FILE HAS NOT BEEN OPENED	File indicated in I/O statement has not been opened via an OPEN statement.
B14	BAD STACK DESCRIPTOR	This error message is generated if the the lengths of the input-lists or lengths of the input-lists or output-lists in the CALL and SUBROUTINE statements are different, if an attempt is made to execute an external subroutine as a main program or if a file variable is used as an operand.
B15	ILLEGAL OPCODE: C	Object code for this item is not legal.
B17	ARRAY SUBSCRIPT OUT-OF-RANGE	Array subscript is less than or equal to zero or exceeds the row or column number indicated by a DIM statement.
B18	ATTRIBUTE LESS THAN -1 IS ILLEGAL	Attribute less than on specified in READV or WRITEV statement.
B25	PROGRAM "C" HAS NOT BEEN CATALOGED	The specified external subroutine must be cataloged before appearing in a CALL statement.
B27	RETURN EXECUTED WITH NO GOSUB	RETURN statement executed prior to GOSUB.
B28	NOT ENOUGH WORK SPACE	Not enough work space assigned at LOGON to run program.
B30	ARRAY SIZE MISMATCH	Array sizes in MAT Copy statement, or in CALL and SUBROUTINE statements, do not match.
B31	STACK OVERFLOW	The program has attempted to call too many nested subroutines.
B32	PAGE HEADING EXCEEDS MAXIUM OF 1400 CHARACTERS	Page heading is too long.
B33	PRECISION DECLARED IN SUBPROGRAM 'C' IS DIFFERENT FROM THAT DECLARED	Precision must be the same between calling programs and subroutines.
B34	FILE VARIABLE USED WHERE STRING EXPRESSION EXPECTED	
B41	LOCK NUMBER IS GREATER THAN 47	Maxium of locks available is 47.

9.127 LIST OF ASCII CODES

This summary presents a list of ASCII codes used in the PICK system.

DECIMAL	HEX	CHARACTER	SPECIAL USE IN PICK
0	0	NUL	Null prompt character
1	1	SOH	Cursor home on CRT Terminal
2	2	STX	
3	3	ETX	
4	4	EOT	
5	5	ENQ	
6	6	ACK	Cursor forward on CRT Terminal
7	7	BEL	Bell on CRT Terminal
8	8	BS	
9	9	HT	
10	A	LF	Cursor down on CRT Terminal
11	B	VT	Vertical address on CRT Terminal
12	C	FF	Screen erase on CRT Terminal
13	D	CR	Carriage return on CRT Terminal
14	E	SO	
15	F	SI	
16	10	DLE	Horizontal address on CRT Terminal
17	11	DC1	
18	12	DC2	
19	13	DC3	
20	14	DC4	
21	15	NAK	Cursor back on CRT Terminal
22	16	SYN	
23	17	ETB	
24	18	CAN	
25	19	EM	
26	1A	SUB	Cursor up on CRT Terminal
27	1B	ESC	
28	1C	FS	
29	1D	GS	
30	1E	RS	
31	1F	US	
32	20	SPACE	
33	21	!	
34	22	"	
35	23	#	
36	24	$	
37	25	%	
38	26	&	
39	27	'	
40	28	(	

41	29	)
42	2A	*
43	2B	+
44	2C	'
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U

86	56	V	
87	57	W	
88	58	X	
89	59	Y	
90	5A	Z	
91	5B	[	
92	5C	/	
93	5D	[	
94	5E		
95	5F	—	
123	7B		
124	7C	:	
125	7D		
126	7E		
127	7F	DEL	
251	FB	SB	Start buffer
252	FC	SVM	Secondary Value Mark
253	FD	VM	Value Mark
254	FE	AM	Attribute Mark
255	FF	SM	Segment Mark

9.128 SUMMARY OF THE PICK/BASIC DEBUGGER COMMANDS

The following is a summary of all the PICK/BASIC DEBUGGER commands and their descriptions.

Bx	Set breakpoint condition table where 'x' is a simple logical expression, which may be composed of < (less than), > (greater than), = (equal to), # (not equal to), & (and), and the special operator $ (line number).
D	Displays breakpoint and trace tables.
DEBUG	Escape to standard debugger.
DE	Short form of DEBUG.
En	Step on n+1 instructions. E <RETURN> turns mode off.
END	End execution of PICK/BASIC program and return to TCL.
G	Proceed from breakpoint.
Gn	Go to line n.
K	Kills all breakpoint conditions in table set by 'B' command.
Kx	Kills breakpoint condition 'x' where 'x' is the breakpoint number from 1-4.
Ln#	Display source code lines starting at n and continuing for # lines.
LP	All output forced to printer reverses status each time LP is selected.
Nx	Continue thru x+1 breakpoints before stopping.
OFF	Log off.
P	Inhibit PICK/BASIC program output.
PC	Printer close - output to spooler.
R	Pops return stack.
S	Display subroutine stack.
T	Turns breakpoint trace table alternatley off and on.
Tv	Set variable 'v' in trace breakpoint table.

U	Remove all breakpoint trace table variables set by 'T' command.
Uv	Remove breakpoint trace variable 'v' from table.
V	Verify object code.
Z	Request symbol table.
$	Current statement number.
?	Display current program name, line # and object code verification status.
/v	Print value of a variable 'v'.
/m(x)	Print value of a point 'x' in array 'm'.
/m(x,y)	Print value of point 'x,y' in array 'm'.
/m	Print the entire array where 'm' is the array.
/*	Dump entire symbol table.
[x,y]	String window where 'x' equals the start of the string and 'y' equals the length of the string. This command effects all outputs of variables and has no effect on input.
[	Removes string window (setting string length to zero has the same effect).
=	Equal sign prints out after the printing of a variable in any slash command except '/m'. The value of the variable may be changed at this point.

NOTE:

- Carriage return terminates all controls.
- A linefeed equals G <RETURN>
- Break key breaks to PICK/BASIC DEBUGGER from PICK/BASIC program at end of line.
- PICK/BASIC DEBUGGER prompts with '*'.

9.129 **APPENDIX G**

BASIC DEBUGGER MESSAGES

The following informative, warning or error messages are used by the BASIC DEBUGGER.

*E x	Single step breakpoint at line number 'x'.
*Bn x	Table breakpoint at line number 'x', 'n' equals number of breakpoint.
*V=x	Value of variable at breakpoint.
*Nvar	Variable not found in statement.
CMND?	Command not recognized.
NSTAT#	Statement number out of range of program.
SYM NOT FND	Symbol not found in table.
UNASSIGNED VAR	Variable not assigned a value.
STACK EMPTY	The subroutine return stack is empty.
STACK ILL	Illegal subroutine return stack format.
TBL FULL	Trace or break table full.
ILLGL SYM	Illegal symbol.
NOT IN TBL	Not in trace break table.
NO SYM TAB	Symbol table not in file.

Chapter 10

SYSTEM MAINTENANCE

THE PICK SYSTEM

USER MANUAL

10 SYSTEM MAINTENANCE

10.1 VIRTUAL MEMORY STRUCTURE 10-3
10.2 ADDITIONAL WORK-SPACE ALLOCATION 10-6
10.3 THE FILE AREA 10-7
10.4 FRAME FORMATS 10-9
10.5 DISPLAYING FRAME FORMATS : THE DUMP VERB 10-10
10.6 THE SYSTEM FILE and SYSTEM-level FILES 10-12
10.7 THE BLOCK-CONVERT AND POINTER-FILE DICTIONARIES . . . 10-14
10.8 THE ERRMSG FILE, LOGON MESSAGES, AND THE PRINT-ERR VERB 10-16
10.9 USER IDENTIFICATION ITEMS 10-19
10.10 SECURITY 10-21
10.11 THE ACCOUNTING HISTORY FILE: AN INTRODUCTION 10-23
10.12 THE ACCOUNTING HISTORY FILE: SUMMARY AND EXAMPLES . . 10-25
10.13 THE ACCOUNTING HISTORY FILE: PERIODIC CLEARING . . . 10-27
10.14 FILE STRUCTURE: THE ITEM AND GROUP COMMANDS 10-28
10.15 FILE STRUCTURE: THE ISTAT AND HASH-TEST COMMANDS . . 10-30
10.16 DETERMINING NATURE OF GROUP FORMAT ERRORS 10-31
10.16.1 GROUP DEFINITION 10-31
10.16.2 GROUP FORMAT ERRORS 10-31
10.16.3 RECOVERY FROM GFE's 10-32
10.17 GENERATING CHECKSUMS: THE CHECK-SUM COMMAND 10-33
10.18 SYSTEM PROGRAMMER (SYSPROG) ACCOUNT 10-34
10.19 AVAILABLE SYSTEM SPACE: THE POVF COMMAND 10-34
10.20 CREATING ACCOUNTS 10-35
10.21 DELETE-ACCOUNT 10-36
10.22 FILE STATISTICS REPORT 10-37
10.23 UTILITY VERBS: STRIP-SOURCE, LOCK-FRAME, UNLOCK-FRAME, 10-39
10.24 SYS-GEN AND FILE-SAVE TAPES: FORMAT 10-41
10.25 FILE-RESTORE 10-42
10.26 ERROR RECOVERY DURING FILE LOADS 10-44
10.27 SELECTIVE RESTORES 10-45
10.28 SYSTEM BACKUP : FILE-SAVE 10-47
10.29 THE SAVE VERB 10-49
10.29.1 MULTIPLE REEL SAVES 10-50
10.30 ACCOUNT-SAVE AND ACCOUNT-RESTORE 10-51
10.31 SYSTEM STATUS: THE WHAT AND WHERE VERBS 10-53
10.32 VERIFYING SOFTWARE 10-56

10.1 VIRTUAL MEMORY STRUCTURE

PICK is a multi-programmable virtual memory machine with all of the virtual memory (i.e., disk) being directly addressable as if it were real memory (i.e., core).

The virtual memory of an PICK system resides on a magnetic disk drive, which is divided into 512 byte "Frames". the frames are given Frame-ID's, or "FID"s numbered 1, 2, 3 ... up to the maximum FID, which depends upon the size of the disk.

The lower-numbered frames on the disk are "ABS" frames, which contain system software and workspaces. all frames above the ABS area are available for use in files. those frames not used in files make up the Available Space, sometimes called "Overflow".

EXECUTABLE AREA (ABS)

The ABS area consists of executable object code ;and process workspaces. Software written in PICK assembly language is loaded onto disk in the executable area. The length of the executable area is a system generation parameter, and must be between 511 and 4095. Frames 1 through 399 of the executable area are reserved for current and future PICK software. The remaining frames are available for user-written assembly language programs.

WORK AREA

The PICK operating system allows multi-programming, which means more than one different program may be executed, on a time-sharing basis, by the CPU. each running program, or process, has a "Primary" workspace area of 32 contiguous frames, the first of which is called the "Process Control Block" (PCB).

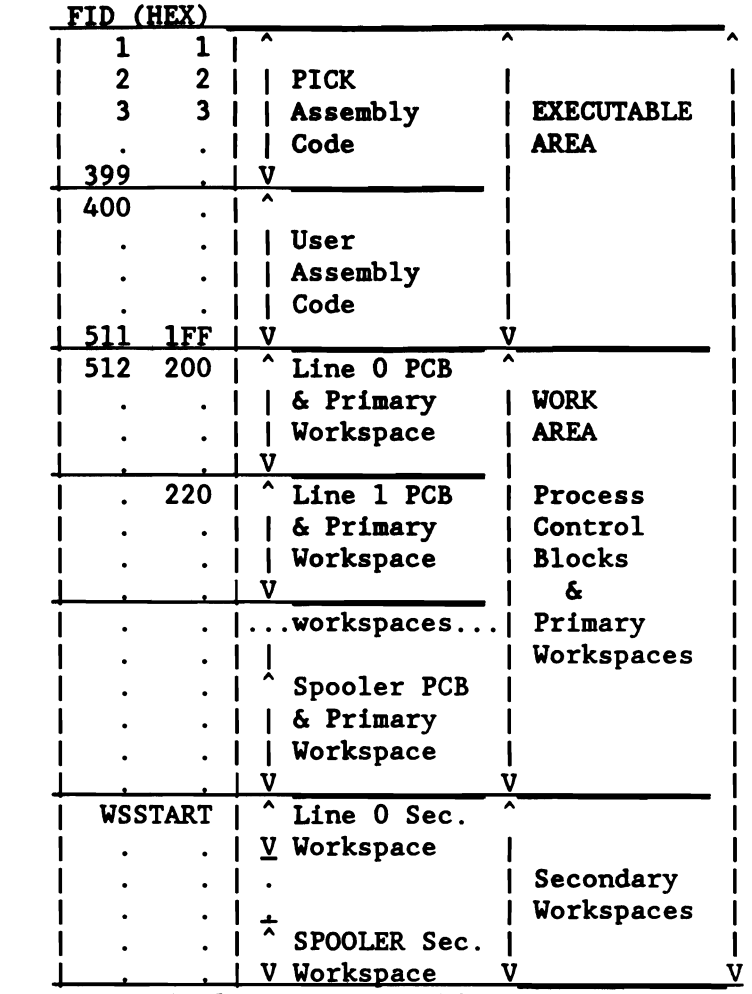
The PCB of channel zero is normally frame 512 (200 hex). PCB's for succeeding processes are separated by 32, and therefore the PCB for channel one is 544 (220 hex), channel two is 240 hex, etc.

Additionally, larger "Secondary" workspace blocks are reserved following the last primary workspace, that of the SPOOLER. WSSTART is the starting FID of the secondary workspaces, which continue to the end of the work area. each process has three secondary workspaces, usually of 100 frames each.

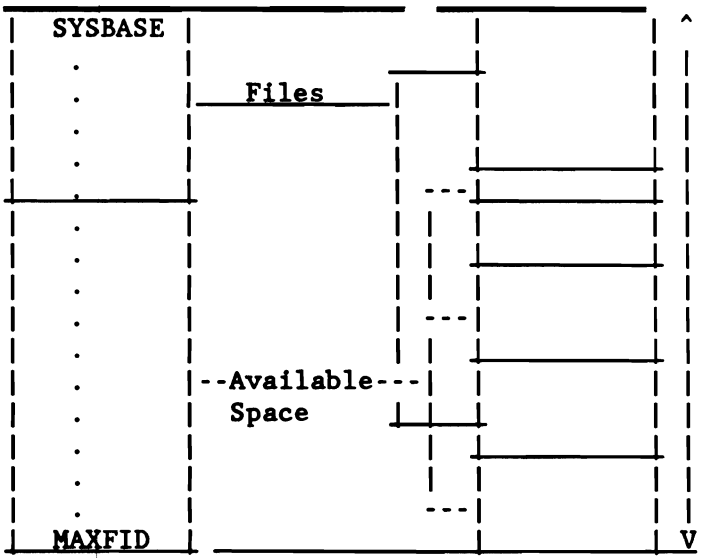
FILES AND OVERFLOW

After the work area are the PICK files, beginning with the SYSTEM file. The base of the SYSTEM file, SYSBASE, is the beginning of the file space. on a newly generated or restored system, all other files on the system immediately follow the SYSTEM file. At the end of the files is the start of Available Space (overflow), which then continues until the end of the disk--MAXFID. (See the left side of the first figure.)

On a running system, the overflow area will become "fragmented", as frames are taken from and returned to the overflow pool. (See the right side of the second figure.)



ABS area, including Executable area and Work area.



Files and Available Space, after a file-restore (left)
and after undergoing normal fragmentation (right)

10.2 ADDITIONAL WORK-SPACE ALLOCATION

The "additional workspace" is a set of contiguous, linked frames that is initialized by the system at coldstart or system-generation time.

There are several processors in the system that require large amounts of workspace, or buffer area. This workspace is pre-assigned, and need not be linked up at LOGON time. The workspace is linked after a file-restore, or it may be linked from TCL by use of the LINK-WS verb. The SPOOLER process links the workspace for all the other lines, and no other user can log on the system while this linkage is taking place; the message:

LINKING WORK-SPACE; WAIT

will appear until the spooler has finished the linkage.

The starting FID of the workspace may be computed as below:

WSSTART = 512 + (number of lines)*32. Each line has three (3) workspaces of one hundred (100) contiguous frames.

The workspace may be linked on a live system using the LINK-WS verb on the SYSPROG account. This may be done if it is suspected that the links of the additional workspace have been destroyed for some reason. One manifestation of this situation is that BASIC programs may terminate with the "NOT ENOUGH WORK SPACE" message. Work-space links should be particularly suspect if a program or process aborts on one channel, but works correctly on others.

The general form of the verb to relink the workspace is:

LINK-WS {(n{-m})}

If the "(n)" or "(n-m)" is omitted, the workspace of ALL lines will be relinked, except those of lines logged on and that of the spooler process. The parenthetical specification may be used to limit the relinking process to lines "n", or "n" through "m" only.

As the linkage proceeds, the line-number of the process whose workspace is currently being linked is displayed on the terminal; if the line is logged on, the message "ON!" will be displayed, and THE WORK-SPACE IS NOT RELINKED!

The spooler's workspace can only be relinked via a coldstart!

10.3 THE FILE AREA

Beginning immediately after the Work Area, the remainder of the virtual memory (called the File Area) is available for the storage of data in files. The portions of the File Area that are not allocated to the files are maintained as a pool of Available Space.

The beginning of the File Area is a system generation parameter. It may be computed via the following general formula:

Start of File Area = (FID of first PCB) +
((number of processes)*32 +
((number of processes)*(pre-assigned work-space)*3

Pre-assigned work-space is set to 100 frames per process per work-space. Each process (including the spooler) has 3 secondary workspaces of 100 frames each.

As an example, a system with 16 communication lines (therefore 17 processes including the spooler) will have the start of the file area at frame:

512 + (17 * 32) + (17 * 300) = 6156

The end of the File Area is the highest available disk frame, MAXFID.

File Area frames which are not allocated to the files are maintained as a pool of Available Space, often called "Overflow". Available Space is used by the Pick system file management routines as additional data space, as well as to other processors as scratch work space. The Pick Computer System maintains a table of pointers that define the Available Space, which may be either in a "linked" form, or in a "contiguous" form. Contiguous Available Space, as the name implies, consists of blocks of contiguous frames (defined by starting and ending numbers) that can be taken out of the pool either singly or as a block. Linked Available Space can only be taken a frame-at-a-time. Conversely, space may be released by processors to the linked available pool a frame-at-a-time, or to the contiguous pool as a block.

At the conclusion of a File-Restore process on the Pick system, an initial condition may be said to exist; there will be one principle block of contiguous available space, extending from the end of the current data space through the last available data frame. This is illustrated in the first figure; the results of the POVF (print overflow) verb indicate that there is no linked overflow space (blank line at the top of the output), and only one contiguous block of space.

As the system obtains and releases Available Space (and as files are created and deleted), the Available Space gets fragmented; at any particular time there may be several blocks of contiguous Available Space, and a chain of linked Available Space. Available frames will be placed in the linked Available Chain only when there are 31 sets of contiguous Available space (representing the maximum that the system space management routines can maintain). This is illustrated in the second figure; here the linked Available chain starts at FID 23459 and contains 400 frames. There are also several sets of contiguous Available space as shown by the pairs of FIDs displayed.

Logically, there is no difference between Available space in linked chain and that in the contiguous sets; however, certain processors obtain frames from the contiguous space only, for example the CREATE-FILE processor, and the MEM-DIAG processor. Therefore, if the system Available space is severely fragmented, while there may be actually be enough disk space to create a large file, for example, there may not be enough available as a contiguous block. Periodically, a File-Restore may be run to restore contiguous Available space from the linked Available space chain.

(SEE: POVF)

```
>POVF [CR]

 23987- 97799

 TOTAL NUMBER OF CONTIGUOUS FRAMES AVAIABLE= 63812
```

Results of POVF immediately after a file-restore
(One contiguous block of Available space only)

```
>POVF [CR]
   23459 (400)
    8112-  8117 (    6)    9000-  9000 (    1)
   23789- 23801 (   13)   25000- 25678 (  679)
   25681- 25692 (   12)   27123- 27323 (  201)
   34502- 35123 (  522)   35800- 35801 (    2)
   37091- 37091 (    1)   37093- 37093 (    1)
   37099- 37100 (    2)   38100- 38100 (    1)
   43100- 44234 ( 1135)   45680- 45681 (    2)
   46343- 46443 (  101)   46445- 46445 (    1)
   46448- 46448 (    1)   46451- 46451 (    1)
   46454- 46454 (    1)   46458- 46474 (   17)
   47011- 47444 (  434)   47460- 47492 (   33)
   47661- 47750 (   90)   48012- 48017 (    6)
   48018- 48018 (    1)   48020- 48101 (   82)
   48233- 48268 (   36)   48299- 48299 (    1)
   51111- 53234 ( 2124)   53400- 53601 (  202)
   60000- 97799 (37800)
  TOTAL NUMBER OF CONTIGUOUS AVAILABLE FRAMES= 43509
```

Results of POVF after normal system operation.

10.4 FRAME FORMATS

A frame is a block of disk space that is referenced by a unique number called the Frame Identifier, or FID. Frames come in two sizes: ABS frames contain are 2048 bytes, file frames contain 512 bytes.

There are two types of frames in the Pick system - ABS frames and FILE frames. ABS frames may be object-code (assembly or PICK/BASIC-compiled object code), buffers, or workspaces required by the system. ABS frames contain 2048 bytes and are not linked.

FILE frames contain 512 bytes; 500 bytes are used for data, the remaining 12 bytes are used as "link fields". Linked frames are used to define data areas that are greater than 1 frame in length. The groups in data files may expand as more data is placed in the group, so when the end of a frame is reached, another frame is obtained from the system overflow and linked to the end of the group.

The format of the linked fields is as follows:

```
byte: 0    1    2    3    4    5    6    7    8    9    A    B    C
      *   nncf  ..forward link...  ..backward link...  npcf  *    start
                                                                  of data
```

where:

*	Unused byte.
nncf	Number of next contiguous frames (count of frames that are linked forwards of this one, whose FID's are sequential to this FID).
npcf	Number of previous contiguous frames (count of frames that are linked backwards to this one, whose FID's are sequential to this FID).
forward link	FID of the frame that is next in logical sequence to this one.
backward link	FID of frame that is logically previous to this one.

The first frame of a linked set of frames will have zero "npcf" and "backward link" fields, and the last frame of such a set will have zero "nncf" and "forward link" fields. The "nncf" and "npcf" fields are also normally zero, except in the "linked workspace" allocated to each process, and in files that have a separation greater than one.

Following the link fields is the 500-byte data block.

Unlinked frames have no specified format; all 512 bytes of the frame may be used by the system.

10.5 DISPLAYING FRAME FORMATS : THE DUMP VERB

The DUMP verb may be used to display data in a frame. The data display may be specified in either character or hexadecimal format.

FORMAT:

```
DUMP n1{-n2},options
```

where

n1{-n2} n1 and n2 specify the FIDs of the frames being dumped; may be specified in decimal, or in hexadecimal by preceding the hex number with a period (.).

options Valid options are

- **C** Display ABS frame; dump begins with byte 0 of the frame and continues for 2048 bytes.
- **G** Group; specifies that the data starting at frame n1 is to be dumped, and that the dump continue following either the forward or backward links (depending on whether the U option is not or is specified). The dump will terminate when the last frame in the logical chain has been found.
- **L** Links; specifies that the dump be confined to the "links" of the frame(s) concerned; no data is displayed.
- **N** No stop; if the data is printed on the terminal, specifies that the end-of-page stop be inhibited.
- **P** Printer; the display is routed to the line-printer.
- **U** Upward trace; the data or links are traced logically upwards, using the backward links to continue the display.
- **X** Hexadecimal display; the frames are dumped in hexadecimal with ASCII characters along the right side of the display.

NOTE: The linkage information displayed by the DUMP verb is meaningful only for linked frames.

```
 >DUMP 6950,L [CR]

 FID:   6950 :   0  6967     0   0  (  1B26 :  0   1B37      0  0 )
+FID:   6967 :   0     0  6950   0  (  1B37 :  0      0   1B26  0 )
```

In the example above, the display indicates that 6950 is the FID whose links are being dumped; the "nncf" field is 0; the "forward link" field is 6967; the "backward link" field is 0; the "npcf" field is 0. Data in parentheses are the same numbers displayed in hexadecimal. The next line displays the link fields of FID 6967; the "+" indicates that this FID is logically "forward" of the preceding one.

```
>DUMP .1DE7 X [CR]

  FID:    7655 :    0   7656      0   0  (  1DE7 :  0   1DE8      0  0
)

0001  8E454E33 32FE2E42 502E462E 4A2E5041   001 :.EN32^.BP.F.J.PA:
0011  52414752 41504820 302E4C45 4654204D   017 :RAGRAPH 0.LEFT M:
0021  41524749 4E203220 2E4C494E 45204C45   033 :ARGIN 2 .LINE LE:
0031  4E475448 203734FE 2E534543 54494F4E   049 :NGTH 74^.SECTION:
0041  2031202E 62662049 53544154 20564552   065 : 1 .bf ISTAT VER:
0051  42202E78 626620FE 2E494E44 45582027   081 :B .xbf ^.INDEX ':
0061  49535441 54207665 726227FE 2E494E44   097 :ISTAT verb'^.IND:
0071  45582027 46696C65 20737461 74697374   113 :EX 'File statist:
0081  69637327 20275374 61746973 74696373   129 :ics' 'Statistics:
0091  206F6620 61206669 6C6527FE 2E626F78   145 : of a file'^.box:
00A1  20FE4953 54415420 69732061 6E204143   161 : ^ISTAT is an AC:
00B1  43455353 20766572 62207768 69636820   177 :CESS verb which :
00C1  70726F76 69646573 2066696C 65207574   193 :provides file ut:
00D1  696C697A 6174696F 6E20696E 666F726D   209 :ilization inform:
00E1  6174696F 6E2EFE2E 78626F78 FE20464F   225 :ation.^.xbox^ FO:
00F1  524D4154 3AFE2E4E 462E494D 203135FE   241 :RMAT:^.NF.IM 15^:
0101  49535441 54207B44 4943547D 2066696C   257 :ISTAT {DICT} fil:
0111  652D6E61 6D65207B 6974656D 2D6C6973   273 :e-name {item-lis:
0121  747D207B 73656C65 6374696F 6E2D6372   289 :t} {selection-cr:
0131  69746572 69617DFE 20202020 20207B6D   305 :iteria}^      {m:
0141  6F646966 69657273 7D207B20 286F7074   321 :odifiers} { (opt:
0151  696F6E73 2C6F7074 696F6E73 2C2E2E2E   337 :ions,options,...:
0161  6F707469 6F6E7329 207DFE2E 494D202D   353 :options) }^.IM -:
0171  31352E46 2E4AFE20 2746696C 652D6E61   369 :15.F.J^ 'File-na:
0181  6D652720 69732074 6865206E 616D6520   385 :me' is the name :
0191  6F662074 68652066 696C6520 666F7220   401 :of the file for :
01A1  77686963 68207468 65207573 6572FE64   417 :which the user^d:
01B1  65736972 65732074 6F207365 65207468   433 :esires to see th:
01C1  65206861 7368696E 67207374 61746973   449 :e hashing statis:
01D1  74696373 2EFE7468 65204953 54415420   465 :tics.^the ISTAT :
01E1  76657262 2070726F 76696465 73206120   481 :verb provides a :
01F1  66696C65                              497 :file:
```

Sample usage of the DUMP verb.

10.6 THE SYSTEM FILE and SYSTEM-level FILES

The SYSTEM file is the highest-level file in the PICK file hierarchy. It contains the file-pointers to every account in the data-base, as well as pointers to the system-level files such as ACC, PROCLIB, etc.

Entries in the SYSTEM file define user M/DICT's or special files neccessary for the PICK software.

The M/DICT pointers are either D (file definition) or Q (file synonym) items. The item-ID's of such items are the USER-NAMES that the user enters when the system requests him to LOGON. Such items are created by the CREATE-ACCOUNT processor, (for D items,) or by use of the EDITOR or COPY processor for Q items. The format of user-identification items is discussed in the section on USER IDENTIFICATION ITEMS.

The SYSTEM file also contains the file-pointers to the system-level files that are necessary to the proper functioning of the system. These files are:

ACC	(Accounting file)
BLOCK-CONVERT	(for BLOCK-PRINT and PICK/BASIC (A) option)
POINTER-FILE	(Saved lists.)
PROCLIB	(Standard system PROC library)
SYSTEM-ERRORS	(Disk errors)

The ACC file (Accounting history) has two types of items: those that indicate the actively logged-on users, and the accounting-history data items that keep track of the usage statistics of each user. The format of the items in this file is discussed in later sections.

The ACC files have a tri-level structure, with an ACC account, an ACC dictionary and an ACC data section.

The BLOCK-CONVERT file contains two unrelated types of items:

1) Items that define the format used in the characters displayed when the BLOCK-PRINT verb is used.

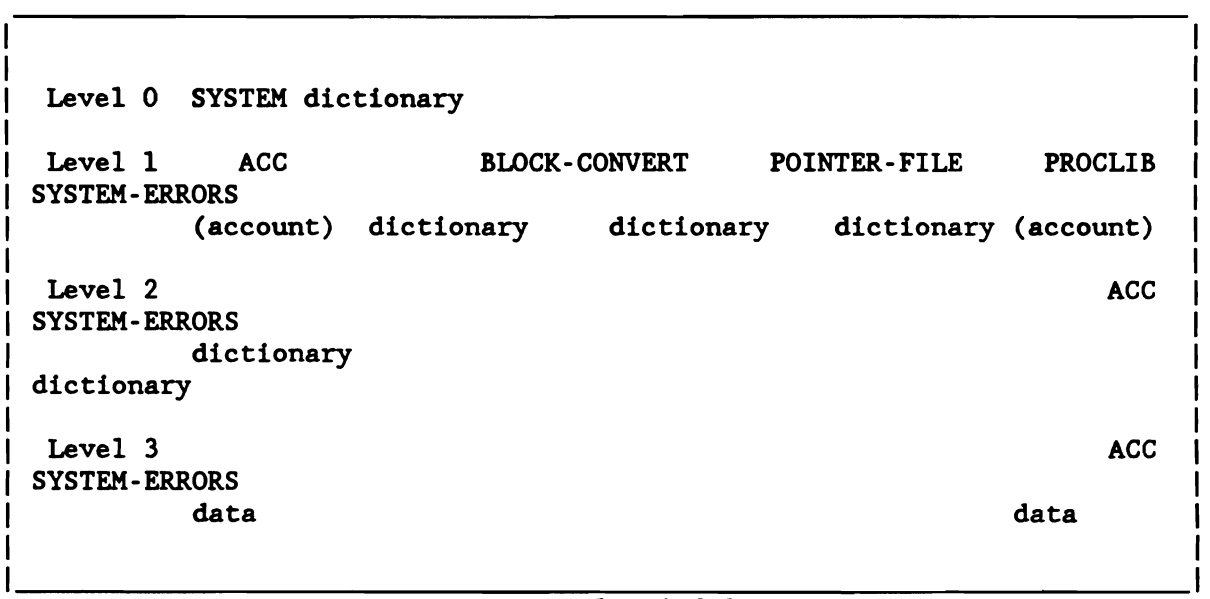
2) Items that are used to print a descriptive message when the "A" (assembly-code) option is used when compiling a PICK/BASIC program.

The BLOCK-CONVERT file is a single-level file.

The POINTER-FILE contains items that are "pointers" to binary data strings. It is referenced implicitly whenever the SAVE-LIST, GET-LIST, DELETE-LIST, CATALOG or DECATALOG verbs are used. The POINTER-FILE is a single-level file. The file-defining entry "POINTER-FILE" in the SYSTEM file must have the code "DC" in line 1. This indicates that the file contains non-standard, binary data items.

The PROCLIB file is a single-level file that contains commonly used PROCs, such as CT (Copy to Terminal), LISTU (List active users), etc.

The SYSTEM-ERRORS file is a three-level file reserved for logging system errors. Currently its only use is to store disk errors.



SYSTEM-level files

10.7 THE BLOCK-CONVERT AND POINTER-FILE DICTIONARIES

This section describes the format of entries in the BLOCK-CONVERT, and POINTER-FILE dictionaries.

BLOCK-CONVERT dictionary:

There are two types of entries in the BLOCK-CONVERT file; type I is the entry that forms the characters for the BLOCK-PRINT verb. Its format is:

Item-id: is the character to be formed; that is, the item whose item-id is "C" will form the character C; that whose item-id is "{" will form the character {, etc.

The first attribute contains a code of the form:

n{c}

where "n" is the width of the character matrix (the depth of all characters formed is fixed at 8); and the optional "c" is a character that will replace the item-id in the generation of the BLOCK-PRINT form.

There must be 8 succeeding attributes, each one specifying the format of a row in the generated form. Each attribute must begin with a "C" or a "B", specifying a character insertion or a blank insertion respectively, followed by the number of such insertions needed; optionally, additional numbers may be specified, separated by commas. Each succeeding number switches the insertion from character to blank and vice-versa. The sum of all numbers must equal the character width specified in attribute 1. See the first example.

The second type of data in the BLOCK-CONVERT file has a two-hexadecimal-digit item-id, corresponding to the PICK/BASIC opcode generated by the PICK/BASIC compiler; attribute 1 is the symbolic name for the opcode. These entries are used by the "A" option of the PICK/BASIC compiler to generate a listing of the PICK/BASIC object code.

POINTER-FILE dictionary:

This file contains the pointers to select-lists (stored by the SAVE-LIST verb) and to cataloged PICK/BASIC programs (stored by the CATALOG verb). They may be examined, but, like file-pointers, **should never be altered in any way by the user!**. The format of these items is:

Item-id	account-name*x*y	where "x" is C for a cataloged program, or "L" for a select-list, and y is the program-name, or select-list name.
001	CL or CC	CL for lists, CC for programs.
002	fid	Base FID of the program or list.
003	n	# frames in the program or list.
004	m	Number of items in a list; null for a program.
005	time & date	Time and date of generation.

```
>COPY BLOCK-CONVERT S 8B (T) [CR]

    S     Item-id; defines format for character "S"
001 7     Defines character width as 7
002 B1,5,1Specifies string " SSSSS " (1 blank, 5 S's, 1 blank)
003 C2,3,2Specifies string "SS   SS"
004 C2,5
005 B1,5,1Specifies string " SSSSS "
006 B5,2
007 C2,3,2
008 B1,5,1
009 B7

    8A    Item-id (BASIC object-code byte)
001 STOP  Identifies object-code (STOP opcode).
```

Sample items from BLOCK-CONVERT file.

```
>BLOCK-PRINT S [CR]

                          SSSSS
                         SS   SS
                         SS
                          SSSSS
                              SS
                         SS   SS
                          SSSSS
```

Output using BLOCK-PRINT verb.

10.8 THE ERRMSG FILE, LOGON MESSAGES, AND THE PRINT-ERR VERB

Most error messages generated by TCL, ACCESS, PICK/BASIC, PROC or any other system software are contained in the ERRMSG file. A standard set of approximately 250 error message items is provided with the PICK base system. However, the user may change the error messages in the ERRMSG file, add new error messages, or even create another ERRMSG file for each account. This can be particularly useful when used in conjunction with the STOP and ABORT statements in PICK/BASIC, in which the user can specify an error message and pass parameters to the error message processor.

Items in the ERRMSG file must follow a certain format, in which the first character in each line of the item defines a special operation, as listed below.

CHARACTER	MEANING
H	Causes the string following the "H" on be placed in the output buffer, with no carriage return or line feed. At the end of the error message item, the string "H+" will inhibit the final carriage-return/line-feed that is normally output.
L	Causes the output buffer to be printed, with a carriage-return and line-feed
L(n)	As above, and also causes n-1 blank lines to be printed.
D	Places the current date in the output buffer.
T	Places the current time in the output buffer.
A	Inserts the next parameter in the list of parameters which was passed to the error message processor with the error message. The parameters may be specified by the PICK/BASIC program (in the case of a PICK/BASIC STOP or ABORT statement,) or by some system processor in the case of system-generated error messages.
R(n)	Inserts the next parameter right-justified in a field of n blanks.
A(n)	Same as R(n), but left-justified.
X	Skips a parameter in the parameter list.
S(n)	Sets the output buffer pointer to location "n".

SPECIAL ERRMSG FILE ITEMS.

The item "LOGON" in the SYSTEM dictionary contains the request to logon to the system (typically "LOGON PLEASE").

When a user logs onto an PICK system, the error message specified by the item "LOGON" in the ERRMSG file is printed on the user's terminal.

Therefore, any message which is to be received by all users on the system immediately upon logging on may be placed in this item. This item must exist on file even if there is to be no general system message.

The ERRMSG items "335" and "336" contain the connect time messages displayed when a user logs on or off the system.

Some examples of error message processing are shown in the first figure.

The PRINT-ERR verb allows the user to invoke the error message processor from TCL. The format is:

>PRINT-ERR file-name item-list

The error messages specified in the item-list will be processed, with a parameter list of A,B,C,D... See second figure.

In a PICK/BASIC program, the lines...

```
FILE = "BP" ; ID = "1006"
OPEN "",FILE ELSE STOP 201,FILE
READ ITEM FROM ID ELSE STOP 202,ID
```

Could cause the program to stop with either of the following:

```
[201] 'BP' IS NOT A FILE NAME
'1006' NOT ON FILE.
```

If the item "LOGON" in the ERRMSG file for an account looked like:

```
HHello out there!
L
HIt's now
T
H and all's well!
```

Then the user would see the following when he logged on:

```
Hello out there!
It's now 11:22:33 and all's well!
```

Sample Usage of the Error Message Processor.

```
>PRINT-ERR ERRMSG 201 [CR]
[201] 'A' IS NOT A FILE NAME

>PRINT-ERR ERRMSG 289

             TERMINAL PRINTER
PAGE WIDTH:      A        B
PAGE DEPTH:      C        D
LINE SKIP :      E
LF DELAY  :      F
FF DELAY  :      G
BACKSPACE :      H
TERM TYPE :      I
```

Sample Usage of the PRINT-ERR verb.

10.9 USER IDENTIFICATION ITEMS

Each user has a user identification item stored in the System Dictionary (SYSTEM). This set of user identification items define the users that can log on to the system. User identification items are either file definition items or file synonym definition items.

User identification items are initially created via the CREATE-ACCOUNT PROC. These items may subsequently be updated via the EDITOR. ENTRIES IN THE SYSTEM DICTIONARY SHOULD NOT BE UPDATED WHEN ANY OTHER USER IS LOGGED ON to the system because the system software maintains pointers to data in the System Dictionary when users log on; updating the System Dictionary invalidates these pointers. An exception to this rule is when creating a new account (or a synonym to an existing account), which can be done at any time since new items are added to the end of the existing System Dictionary data, and thus do not disturb any pointers to it.

Attributes five through eight of a user identification item contain data associated with the user's security (lock) codes, password, and system privileges:

ATTRIBUTE	USE
5	Contains the set of retrieval lock-codes associated with the user; may be multi-valued (separated by value marks). There is no restriction to the format of individual lock-codes. This attribute may be null, indicating no lock-codes. (Lock-code usage is described in the topic titled SECURITY.)
6	Contains the set of update lock-codes associated with the user; the same considerations as described for retrieval lock-codes above.
7	Contains the user's password, which is a single value. This attribute may be null. This field is encoded by the PASSWORD process and should be changed only by the PASSWORD verb.
8	Contains a code which indicates the level of system privileges (see below) assigned to the user.
9	May contain the code "U" to indicate that logon/logoff times should be logged by the system. May contain the code "R" to specify the RESTART option.

Attributes one through four and attributes ten through thirteen are as defined for regular file definition of file synonym definition items (see topic titled DICTIONARIES). The first figure shows a sample user identification item for user SMITH.

Attribute 8 contains the system privilege level. Three levels are available; they are referred to as SYS0 (lowest), SYS1, and SYS2 (highest), respectively. Lower levels of system privileges restrict usage of certain facilities of the system, as described in the second figure.

Attribute 9 may contain the codes 'U' or 'R', or both. 'U' specifies that the accounting listing file is to be updated whenever the user logs on and off this account (see ACCOUNTING FILE). 'R' specifies that the Restart option is to be set. This causes the LOGON PROC to be re-executed whenever an "END" is typed at the DEBUG level.

item SMITH in System Dictionary

```
001 D <------------------------- D/CODE
002 2537 <---------------------- Base FID
003 13 <------------------------ Modulo
004 1 <------------------------- Separation
005 ABC <----------------------- Retrieval lock Code (L/RET)
006 1234 <---------------------- Update Lock Code (L/UPD)
007 PW5 <----------------------- Password
008 SYS2 <---------------------- System Privilege Level
009 U <------------------------- Update Account File for this user
```

Sample User Identification Item For User SMITH

FACILITY	LOWEST PRIVILEGE LEVEL REQUIRED
Updating of M/DICT	One
Use of magnetic tape	One
Use of DEBUG (other than P, OFF, END and G commands).	Two
Use of DUMP Processor	Two
Use of Assembler and Loader	Two
Use of FILE-SAVE and FILE-RESTORE processors.	Two

Required System Privilege Levels.

Security codes may optionally be placed in the L/RET and L/UPD attributes of a dictionary item to restrict access and update. At Logon time, each user is assigned the set of security codes which are in his user identification item. During the session, whenever, an L/RET or L/UPD code is encountered, a search is made of the user assigned codes for a match; if no match is found the user is denied access. A security code may consist of any combination of legal ASCII characters.

L/RET and L/UPD

Both file definition ("D" code) and synonym file definition ("Q" code) items have L/RET (retrieval locks) and L/UPD (update locks) attributes. When these attributes have values stored, they are known as security codes. Although there is no prohibition against multiple values for these attributes, only the first attribute value is used for matching against the user assigned codes. Since each file may be individually locked for both update and retrieval, a user must be assigned multiple codes to that set of data he is allowed to access. Using this feature, a complex "mask" can be constructed for each user, giving each user a different sub-set of files which he may access.

Security at the file level is invoked at the processor level. The following processors are assumed to be updating processors and therefore require a match on the L/UPD attribute in the file definition item: COPY, EDIT, PICK/BASIC if writing a file, RUN and the Assembler. Other processors are assumed to be retrieval processors and require a match on the L/RET attribute in the file definition item.

PICK/BASIC requires a match against L/RET code when the file is opened; and requires a match against the L/UPD if data is changed in the file.

Failure to match one of the user security codes with either the L/RET (or L/UPD) attribute value will generate the following message (and return control to TCL):

```
[210] FILE xxx IS ACCESS PROTECTED
```

User Assigned Codes

Each user identification item in the System Dictionary (see topic titled USER IDENTIFICATION ITEMS) contains the list of security codes assigned for that particular user. These codes are values for the attributes L/RET and L/UPD. The lock code in the user-identification item and the lock code in the file being verified must match.

Security codes may be assigned initially when an account is created via use of the CREATE-ACCOUNT PROC Security codes may be added or deleted by updating the appropriate security codes); however, updates to the user identification item should only be performed when no one else is logged onto the system.

Security Code Comparison

Security codes are verified comparing the value in the file dictionary against the corresponding string of values in the user identification item. Characters are compared from left to right. An equal (verified) compare occurs when the value in the file dictionary is exhausted and all characters match up to that point. This is illustrated below.

When referencing a file using a Q synonym a security code match is made at all levels (i.e., SYSTEM, M/DICT, and file dictionary) and therefore a correspondence must be maintained at all levels in order to process the Q synonym files. Since the user identification item for the account containing the primary file is verified for security codes, the user referencing the Q synonym must have a code defined in this user identification item which will verify with the first code in the equated account's user identification item. Therefore, in a user identification item, only the first code is used to protect the account from Q synonym accesses,while all the codes in the item are assigned to the user when he logs on.

FILE DICTIONARY CODE	USER IDENTIFICATION CODE	RESULT
123	123	Match
12	123	Match
123	12	No Match
XYZ	XYZ5	Match
AQ2	AQ	No Match

Sample Security Code Comparisons.

10.11 THE ACCOUNTING HISTORY FILE: AN INTRODUCTION

The Accounting History File is one of the mandatory files in the PICK system. This file contains accounting history for the system, as well as the entries that describe the currently active (logged-on) users.

The System dictionary (SYSTEM) contains the file definition item (item-id 'ACC') for the Accounting History File, as illustrated in the figure. The 'ACC' dictionary is set up for examining and listing the data in Accounting History File via ACCESS (see topic titled THE ACCOUNTING HISTORY FILE: SUMMARY AND EXAMPLES). There are two types of entries (items) in the Accounting History File: those that represent active (logged-on) users, and those that keep track of accounting history.

Active Users Items

The item-id of an active user item in the Accounting History File is the four-character hexadecimal FID of the PCB of the user's process. If the PCB's start at FID=512, (they proceed in steps of 32 frames from there on), we see that a user logged on to process zero will have an entry with an item-id '0200' (512), while a user logged on to process one will have an entry with an item-id '0220' (544), and so on. Attribute one of an active user item contains the name of the user (i.e., the item-id of the user identification item), attribute three the date logged on, and attribute four the time logged on. Active user items are created when a user logs on, and deleted when he logs off.

Accounting History Items

The item-id of an accounting history item is the name of the user (i.e., the item-id of the user identification item), with the channel number concatenated by a "#". For example, if user 'SMITH' logs on to channel 12, when he logs off, the item whose item-id is 'SMITH#12' in the ACC file will be updated. This allows one to keep track of system usage by user-id as well as channel number.

Attributes one, two and three are not used. The remainder of the attributes are described below:

ATTRIBUTE	USE
4	Dates(s) Logged on. Each unique date is stored. Value marks are tagged on to the value in this attribute if multiple Logoffs occur on the same date (for LIST alignment purposes). Date is stored in Pick Computer System date format.
5	Time(s) Logged on. An entry is made for each Log-off, representing the time at which the user Logged on. Time is represented in seconds past midnight (24- hour clock).
6	Connect time(s). This entry represents the time in seconds between the Logon and the Logoff.

7 Charge-units. A number representing the CPU usage is added on each Logoff.

8 Line-printer pages. A number representing the number of pages routed to the line-printer for each session.

Note: Attributes 4, 5, 6, 7 and 8 are stored as a "Controlling-dependent" data set, with attribute 4 being the controlling value, and the others the dependent ones. See the ACCESS reference manual for a discussion of the "controlling-dependent" data set format.

The accounting history file 'ACC' is not automatically updated every time a user logs off the system. The SYSTEM dictionary item for the user must have a 'U' in attribute 9 if the user is to have his Account file history items updated. The entries in the Account file contain the history of each session (logon to logoff). If the SYSTEM dictionary data has been changed since logon or the history item to the updated is too large for the work-space, the message number 338 will be printed.

Channel #	Item-id	Channel #	Item-id
0	0200	16	0400
1	0220	17	0420
2	0240	18	0440
3	0260	19	0460
4	0280	20	0480
5	02A0	21	04A0
6	02C0	22	04C0
7	02E0	23	04E0
8	0300	24	0500
9	0320	25	0520
10	0340	26	0540
11	0360	27	0560
12	0380	28	0580
13	03A0	29	05A0
14	03C0	30	05C0
15	03E0		

Channel (Line) numbers and corrosponding Active User Item-IDs.

10.12 THE ACCOUNTING HISTORY FILE: SUMMARY AND EXAMPLES

This topic summarizes the formats of the active user items and the accounting history items in the Accounting History File. Also presented are sample entries for the Accounting History File.

The first figure summarizes the attributes for the active user items and the accounting history items. The second figure shows a sample sorted listing of the active users (users with a value for attribute A1) via an ACCESS SORT statement. The third figure shows a sample listing of the accounting history item for user SMITH via an ACCESS LIST statement.

ATTRIBUTE NUMBER	'ACC' DICTIONARY NAME	ACTIVE USER ITEM	ACCOUNTING HISTORY ITEM
	(item-id)	Four-character hexadecimal PCB-FID	User name#lineno
1	NAME	User name	Not used
2	DATE	Date logged on	Not used
3	TIME	Time logged on	Not used
4	DATES		Dates logged on
5	TIMES		Times logged on
6	CONN		Connect time
7	UNITS		Charge-units
8	PAGES		Number of printer pages generated.

Summary of Active User Items and Accounting History Items

```
>LISTU [CR]

CH# PCBF NAME.......... TIME... DATE.... LOCATION.....................

 00 0200 CM             11:02AM 03/22/78 Channel 0
 01 0220 SYSPROG        12:10PM 03/22/78 Channel 1
 02 0240 EL-ROD         09:11AM 03/22/78 Channel 2
 03 0260 LC             06:59AM 03/22/78 Channel 3
 05 02A0 HVE            09:55AM 03/22/78 Channel 5
*06 02C0 CM             11:25AM 03/22/78 Channel 6
 07 02E0 BUGEYE         01:29PM 03/21/78 Channel 7
 10 0340 JT             11:34AM 03/22/78 Channel 10
```

Sample sorted listing of active user items (using LISTU).

```
>LIST ACC = "SMITH]"                (selects items with item-ids
                                      starting with the string "SMITH")
PAGE   1                                        12:17:22  22 MAR 1978

ACC..... DATE. TIME... CONN... UNITS.. PAGES
               *       *       *       *
SMITH#0  01/13   16:56   00:04       9
         01/14   10:13   00:00       5
                 10:15   00:01     343
         02/06   17:02   00:18      41
         02/09   10:21   00:17     690
         02/23   07:58   00:01      27
         03/09   11:35   01:57     378
                 16:05   00:22      94
SMITH#5  01/13   12:48   02:25     160      5
                 15:20   00:05      14
                 15:25   00:00       2
                 15:28   00:17     110
                 16:20   02:55    2575     16
                 19:15   00:00      13
         01/16   09:41   06:13    1853      6
                 15:55   00:12      15

2 ITEMS LISTED.
```

Sample listing of accounting-history item for user "SMITH".

10.13 THE ACCOUNTING HISTORY FILE: PERIODIC CLEARING

To avoid overflowing the accounting history items in the Accounting History file for a specific user, the items should be periodically cleared.

To clear the accounting history items from the ACC file, follow the steps detailed in the first figure.

The point of overflow is determined by the activity of the user-account (however, approximately 1000 Logon/Logoffs are allowed). This point can be calculated by following the procedure detailed in the second figure.

If the accounting history item for a user-account does exceed the available workspace, the user will be logged off, but the Accounting History File will not be updated. To recover from this situation, follow the procedure detailed below.

1. Logon to the SYSPROG account.

2. Type the following (if you need a listing only):

```
>SORT ACC WITH NAME LPTR [CR]
```

3. Type the following:

```
>SELECT ACC WITH NAME [CR]
>DELETE ACC  [CR]
```

Procedure to Clear all Accounting History Items.

1. Use the WHAT verb to determine the number of additional work-space frames allocated for the system (parameter WSSIZE in the WHAT display). Multiply this figure by 500 and add 3000.

2. To determine the current size, type:

```
>STAT ACC ACC-SIZE 'user-name'  [CR]
```

This will produce the following output:

```
STATISTICS OF ACC-SIZE:
TOTAL = xxx AVERAGE = yyy COUNT = zzz
```

3. If the value displayed for TOTAL in step 2 (i.e., xxx) approaches the value calculated in step 1, then the user-account is approaching the overflow point.

Determining the point of overflow for an accounting-history item.

10.14 FILE STRUCTURE: THE ITEM AND GROUP COMMANDS

The ITEM and GROUP commands provide information about the item and group structure of Pick files. Output can be displayed at the terminal or optionally directed to the line printer.

FORMAT:

ITEM file-name item-id {(options)}

This command displays the base FID of the group into which the specified item-id hashes. If the item is not already on file, the message "ITEM NOT FOUND" is displayed. In addition, every item-id in that group is listed along with a character count of the item (in hex). At the end of the list the following message is displayed:

n ITEMS m BYTES p/q FRAMES

where:

n is the number of items in the group.
m is the total number of bytes used in the group.
p is the number of full frames in the group.
q is the number of bytes used in the last frame of the group.

Valid options for this command are as follows:

P - Direct output to line printer.
S - Suppress item list.

FORMAT:

GROUP file-name {(options)}

This command displays the base FID of each group in the specified file. In addition, every item-id in the group is listed along with a character count of the item (in hex). At the end of the list for each group the following message is displayed:

n ITEMS m BYTES p/q FRAMES

where:

n is the number of items in the group.
m is the total number bytes used in the group.
p is the number of full frames in the group.
q is the number of bytes used in the last frame of the group.

Valid options for this command are as follows:

P - Direct output to line printer.
S - Suppress item list.

```
>ITEM M/DICT A [CR]

27121
0022 FILE-DOC
001C bd
0009 A
0011 T-ATT
000F DUMP
0018 B/ADD
000F DIVX
0014 EDIT-LIST
0028 V/CONV
0022 LISTU
0019 V/MIN
001B ACCOUNT-RESTORE
001D D/CODE
0028 SL
0023 INST-INDEX
0047 SAL
0072 TB
000E SAVE
18 ITEMS 591 BYTES 1/91 FRAMES
```

Displaying data in a group using the ITEM command.

10.15 FILE STRUCTURE: THE ISTAT AND HASH-TEST COMMANDS

ISTAT and HASH-TEST are ACCESS verbs that produce file hashing histograms, ISTAT for specified file items and HASH-TEST on the basis of a user-specified test modulo.

ISTAT

An ACCESS sentence using the ISTAT command is constructed as illustrated below. The ISTAT command provides a file hashing histogram for the selected items in the selected file, as illustrated by the examples. For further information regarding file hashing, refer to the section of this manual titled VIRTUAL MEMORY OPERATING SYSTEM. HASH-TEST

HASH-TEST produces a file hashing histogram as a result of a user-specified test modulo. The general form of this verb is as follows:

```
HASH-TEST (DICT) file-name (item-list) (selection-criteria)
```

```
>ISTAT PARCEL [CR]

FILE= PARCEL MODULO= 3 SEPAR= 1                13:50:42  22 MAR 1978
FRAMES BYTES ITMS
000002 00757 027 *>>>>>>>>>>>>>>>>>>>>>>>>>>>
000002 00836 030 *>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
000002 00785 028 *>>>>>>>>>>>>>>>>>>>>>>>>>>>

ITEM COUNT=          85, BYTE COUNT=      2378, AVG. BYTES/ITEM=     27.9
AVG. ITEMS/GROUP=  28.3, STD. DEVIATION=   1.5, AVG. BYTES/GROUP=   792.6.
```

Sample usage of the ISTAT command.

```
>HASH-TEST PARCEL [CR]

TEST MODULO: 9 [CR]
FILE= PARCEL MODULO= 9 SEPAR= 1                13:50:55  22 MAR 1978
FRAMES BYTES ITMS
000001 00256 009 *>>>>>>>>>
000001 00281 010 *>>>>>>>>>>
000001 00255 009 *>>>>>>>>>
000001 00229 008 *>>>>>>>>
000001 00248 009 *>>>>>>>>>
000001 00251 009 *>>>>>>>>>
000001 00272 010 *>>>>>>>>>>
000001 00307 011 *>>>>>>>>>>>
000001 00279 010 *>>>>>>>>>>

ITEM COUNT=          85, BYTE COUNT=      2378, AVG. BYTES/ITEM=     27.9
AVG. ITEMS/GROUP=   9.4, STD. DEVIATION=    .8, AVG. BYTES/GROUP=   264.2.
```

Sample usage of the HASH-TEST verb.

10.16 DETERMINING NATURE OF GROUP FORMAT ERRORS

10.16.1 GROUP DEFINITION

The term group is used to specify one 'bucket' of storage. A file is made up of a collection of groups, such that there are the same number of groups as the number specified for the modulo of the file. Put another way, the modulo of the file specifies the number of groups which make up the file.

The hashing algorithm takes the specified item-id and decides in which group it is or should be stored. The file retrieval or storage routine then searches that group for the specified item. The hashing algorithm may be thought of as dividing the item-id by the modulo in order to obtain the remainder. This remainder is then the 'group number', and specifies the group which is to be searched.

Within each group the items are stored physically end to end. Each item is made up of a count field, a key, and the data. The documentation for this system has conventionally used the term 'item-id' in place of the term 'key'. It remains that the item-id is the key which is used to look up the location of the item.

The count field exists only in a file representation of the item. It is a sixteen-bit binary number, such that the high-order bit is zero, represented in the file in ASCII hexadecimal notation, and as such takes up four bytes of storage. It immediately precedes the item-id in the file. If the item in question is the first item in the group, the count field starts in the first data byte in the frame. If the item is not the first item in the group, then the count field starts at the first byte after the termination mark of the last item.

The count field is used as a pointer to the end of the item. The end of the item must be an attribute mark followed by a segment mark. If the count field does not point to this pattern, there is a group format error, and the group format error handler will be entered.

10.16.2 GROUP FORMAT ERRORS A GROUP FORMAT ERROR IS THE RESULT OF A HARDWARE ERROR!

A group format error is sensed if the count field does not point at an attribute mark, segment mark sequence. This may occur if the count is wrong, or if the data at the end of the item is wrong.

The count field is definitely wrong if any of the four digits which make up the count field are not ASCII hexadecimal digits, which are X'30' - X'39' or X'41' - X'46', which are 0-9 and A-F.

The end of item data may be wrong if the count field contains the wrong ASCII hexadecimal digits, or if the end of item data is actually wrong.

The end of item data may be wrong in several ways. If the item is contained in a frame, then the end of item data may be wrong in the ways that the the count field may be wrong. If the item spans a frame boundary, certain other mechanisms come into play. If a process was in the process of updating an item, to the extent that the first frame containing the item was written to disk, but that the last frame was not written when the process was interrupted by something like a cold start, then a group format error will occur. If the overflow handler becomes confused, the frames attached to a group may be acquired by another data file or by a print file. The difference should be obvious on inspection, using the DUMP verb. Print files do not normally contain attribute or value marks and data files do not normally contain carraige-return, line-feed sequences.

If the damaged frame is the result of an incomplete update, then the difficulty is localized. Repair of this group will usually attend to the matter. If the damage appears to be due to co-ownership of the frame, the problem may be greater. In this case it is best to leave the frame with the frame to which it has a back-link, presuming that the data is consistent in that chain. Then cut the forward link in the spurious chain and terminate the group.

The effect of the group format error handler is to terminate the group at the end of the last consistent item and cut the forward link out of the last acceptable frame in the group. The rest of the overflow is intentionally lost, because of the effect of having two copies of the same frame referenced in the overflow chain.

The one case in which the group will not be terminated is when a print file has meandered across the base of the file. In this case it is probably best to recreate the file and selectively restore it. The old file pointer should be thrown away. Do not use the DELETE-FILE verb on the old file, because this will further muddy the condition of the overflow handler.

10.16.3 RECOVERY FROM GFE's

If a group format error is encountered, the system will invoke the group format error handler. This processor will print the error message to the terminal and wait for an operator response. The valid operator responses are:

'D' - which will enter the system debugger.

'E' - which will end the process and return to TCL.

'F' - which will allow the GFE handler to fix the error and continue.

NOTE that fixing the error will undoubtedly cause the loss of at least one data item. This record normally must be manually recovered! The recovery strategy is to identify the file affected and do a SEL- RESTORE on the file. It is best to do this as soon after the group format error is noticed as possible.

10.17 GENERATING CHECKSUMS: THE CHECK-SUM COMMAND

The CHECK-SUM command generates a checksum for file items, thus providing a means to determine if data in a file has been changed.

FORMAT:

CHECK-SUM (DICT) file-name (item-list) (attribute) (selection-criteria)

A checksum is generated for items in the specified file, or subset of items if the optional "item-list" and/or "selection-criteria" appear. Furthermore, the checksum may be calculated for one specified attribute. If no attribute is specified, the 1st default attribute will be used. If there is no default attribute, or if the AMC is 9999, the entire item will be included. The checksum will include the binary value of each character times a positional value. This yields a checksum which has a high probability of being unique for a given character string. The dictionary portion is checksummed if the "DICT" option appears. (A checksum is the arithmetic total, disregarding overflow, of all bytes in the selected items.)

A message is output, giving checksum statistics, in the following form:

```
        BYTE STATISTICS FOR  file-name (or attribute name):
   TOTAL = t AVERAGE = a ITEMS = i CKSUM= c BITS = b
```

where:

t is the total number of bytes in the attribute (or item) included
a is the average number of bytes
i is the number of items
c is the checksum
b is a bit count

The attribute mark trailing the specified attribute (or item) will be included in the statistics.

To use checksums, the user should issue CHECK-SUM commands for all files, or portions of files, to be verified and keep the output statistics. Subsequently, the CHECK-SUM commands can be reissued to verify that the checksum statistics have not changed. The checksum must be recalculated whenever the user updates the file!

10.18 SYSTEM PROGRAMMER (SYSPROG) ACCOUNT

Several special faciities are normally used from the System Programmer (SYSPROG) Account. Procedures on this account are normally performed by persons more familiar with the overall operation of the system.

To log on to the SYSPROG Account, type the following:

LOGON PLEASE: SYSPROG,password [CR]

where "password" is the appropriate password set up for SYSPROG. Alternate logon names (such as SP) may be used.

CREATE-ACCOUNT	DELETE-ACCOUNT
ACCOUNT-RESTORE	SAVE
BUFFERS	SEL-RESTORE
LOCK-FRAME	UNLOCK-FRAME
:FILES	:ABSLOAD
:ABS/FILES	WHAT

Some SYSPROG Verbs and Procs.

10.19 AVAILABLE SYSTEM SPACE: THE POVF COMMAND

The POVF verb displays the system overflow table.

FORMAT:

POVF {(P}

The POVF verb displays the contents of the system overflow table.

The P option forces all printed output to the line printer. the first line of output is the FID of the first frame in linked overflow, folowed by the number of frames in the linked chain. the next lines (up to 16) describe blocks of contiguous overflow, and have the following format:

m - n : p m - n : p

where:

m is the first frame of a contiguous block.
n is the last frame of the block.
p is the number of frames in the block.

The total number of frames contained in all the contiguous overflow is then printed (using error message number 293):

TOTAL NUMBER OF CONTIGUOUS FRAMES :number

The CREATE-ACCOUNT PROC is used to create new user-accounts.

CREATE-ACCOUNT PROC

The CREATE-ACCOUNT PROC generates a new account according to given specifications. It then copies the contents of the NEWAC file (the prototype M/DICT) to the new user M/DICT. Finally, it adds a file synonym (Q item) to the account into SYSPROG's M/DICT. The CREATE-ACCOUNT PROC is invoked by typing in the PROC name:

```
>CREATE-ACCOUNT  [CR]
```

The PROC then prompts the user for the required information, as shown below.

NOTE: The CREATE-ACCOUNT PROC should not be used to create a new synonym to an existing account; this should be done by using the EDITOR to create the file synonym definition item (Q-item) in the SYSTEM dictionary.

```
>CREATE-ACCOUNT          PROC is typed in at TCL.
ACCOUNT NAME?SHERRY      Anything but [CR] is legal.
L/RET-CODE(S)?AAA]BBB    Multi-valued retrieval code.
L/UPD-CODE(S)?           [CR] means no lock code.
PRIVILEGES?              [CR] means SYS0. May be SYS0, SYS1, or SYS2.
MOD, SEP?37,1            [CR] defaults to 29,1.

CREATE-FILE (DICT SHERRY 37,1

[417] FILE 'SHERRY' CREATED; BASE= 34593 MODULO= 37 SEPAR = 1.

280 ITEMS COPIED
'SHERRY' ADDED
'SHERRY' UPDATED
PASSWORD?R2D2            User's LOGON password.

FINISHED
```

Sample CREATE-ACCOUNT Usage.

10.21 DELETE-ACCOUNT

DELETE-ACCOUNT deletes an account and all its files from an PICK system.

DELETE-ACCOUNT is a PROC which runs the program DEL-ACC. The program lists all the files in the specified account, then requests verification to delete the account. The files may be listed on the terminal or the printer.

Requirements to run DELETE-ACCOUNT:

1. You must be logged on to SYSPROG.
2. SYSPROG must have Q-pointers to the MD of the account, and to SYSTEM.
3. D-items must exist in DICT SYSTEM for SYSPROG and the account name.
4. SYSPROG must have access to SYSTEM and all files on the account to be deleted.

ALL USERS SHOULD LOG OFF before running this because an item in the SYSTEM dictionary will be deleted.

```
>DELETE-ACCOUNT                    PROC name

Account Name ?SHERRY               Enter account name

List Files on Printer (Y/N) ?      To list files on printer, enter Y.

Files to be Deleted in Account:  SHERRY  11:29:14  02 APR 88  PAGE   1

FILE      Type      BASE     MOD  SEP

GEN/LED   D         85344    1    1
GEN/LED   D         49911    231  1
BP        D         44319    17   5

Still want to Delete Account SHERRY? Y
                                   To delete the account, enter Y.
```

Sample DELETE-ACCOUNT usage.

10.22 FILE STATISTICS REPORT

The File Statistics Report provides a list of file parameters, such as name, base, modulo, and file size. It also provides the order of files on a FILE-SAVE tape. The report is automatically generated by running a FILE-SAVE, or may be generated at any time by using the PROC LIST-FILE-STATS.

The report is broken down by account, with a line of information generated for each file in the account that includes:

item.id
name
base, modulo, and separation
total file size
total number of frames used
utilization of file space
number of Group Format Errors (GFEs)

If the report is being sent to a printer that prints 132 columns, the following additional information is included:

average item size
average number of items per group
pad space; that is, unused space

A total line is generated for each account.

The information for the report is kept in the STAT-FILE on the account that does the FILE-SAVE; this is usually the SYSPROG account. The FILE-SAVE process creates one item in the STAT-FILE for each file saved on the file-save tape. The item-ids in the STAT-FILE are of the form

t:n

where

t tape reel number where the file was dumped (this is 0 if the SAVE was run without dumping data to the tape)

n file number; this file-number is used in the selective restoration of files using SEL-RESTORE.

The NAME field of the items in the STAT-FILE contains data in the form:

dictname (dictionary file)
dictname*dataname (data file)

When a FILE-SAVE is started, the STAT-FILE data area is cleared and the current file statistics information is written into the data area. The STAT-FILE data area is also empty after a file-restore is done, because attribute 1 of the file definition is a DY. This is desirable as the statistics are no longer applicable.


```
ITM-ID R#  ID NAME..... BASE. MOD S ...SIZE FRAMES %UT* GFE

1:367  1  367 ACC          9253   1 1      29      1   5
1:368  1  368 ACC          9254   1 1   2,191      5  87
1:369  1  369 ACC*ACC       9255  13 1      54     13   0
                                        ------    --- ---   ---
*** Totals for user:  ACC                2,274     19  23

1:2    1    2 BLOCK-CONVERT    8149  11 1   8,324     24  69
     .
     .

*The utilization of file space is derived by dividing the size  of the
 file by the number of frames.
```

10.23 UTILITY VERBS: STRIP-SOURCE, LOCK-FRAME, UNLOCK-FRAME, CHARGES, AND CHARGE-TO

This topic describes a number of special utility verbs.

STRIP-SOURCE Verb

The STRIP-SOURCE verb is a TCL-II verb used to remove the source code from Assembly Language programs. This frees large amounts of disk space back to the available space pool. Modes with source stripped out can still be verified against the ABS.

FORMAT:

STRIP-SOURCE file-name item-list

After the verb has been invoked, the user is prompted with:

DESTINATION FILE:

The file-name where the stripped object code is to be stored should then be entered. For example:

```
>STRIP-SOURCE PROG * [CR]
DESTINATION FILE-SPROG [CR]
```

Here the file PROG containing source programs is stripped and copied to the file SPROG.

The first six lines of the source item will be copied without source code stripping. Standard Pick Systems convention for source modes has the "FRAME" statement in line 1, and other descriptive information in lines 2 through 6; this information is maintained through the STRIP-SOURCE process.

LOCK-FRAME Verb

The LOCK-FRAME verb may be used to core lock a frame.

FORMAT:

LOCK-FRAME number

where "number" is a decimal frame number. The LOCK-FRAME verb responds with the absolute hexadecimal work address of the memory buffer in which the frame is corelocked. The frame remains corelocked until it is released by the UNLOCK-FRAME verb, or the system is re-booted.

UNLOCK-FRAME Verb

The UNLOCK-FRAME verb clears the corelocked buffer status of the frame indicated.

FORMAT:

UNLOCK-FRAME number

where "number" is a decimal frame number.

CHARGES Verb

The CHARGES verb prints the current computer usage since logon as connect time in minutes and CPU usage in charge-units.

FORMAT:

CHARGES

CHARGE-TO Verb

The CHARGE-TO verb is used to keep track of computer usage for several projects associated with the same logon name.

FORMAT:

CHARGE-TO name

This verb performs the following:

1. Terminates the current charge session by updating the ACC file with the user's accumulated charge-units, line printer pages and connect-time statistics.

2. Changes the logon name to the original name concatenated with an asterisk and then the name following "CHARGE-TO".

For example, if the user is currently logged on to SYSPROG, and he types in the following:

```
>CHARGE-TO PROJECT1  [CR]
```

the LOGON name in the ACC file for the process will be changed to "SYSPROG*PROJECT1".

10.24 SYS-GEN AND FILE-SAVE TAPES: FORMAT

System restore is the process of "bringing up", or creating, the PICK system. A bootstrap program, all system software, and all files can be loaded from magnetic tape. The system configuration is set up at cold-start time.

A system can be restored from a SYS-GEN tape or a file-save tape. There are three sections on a SYS-GEN tape:

1. The bootstrap section contains the MONITOR, the configurator, and some virtual program frames needed to bootstrap the system. There are 33 tape records in this section, followed by an END-OF-FILE mark (EOF).

2. The ABS section, which contains the system software. This section is preceded by a tape label, which contains the release level, and terminated by an EOF. This software makes up the PICK Operating System, the Language processors (ACCESS, PICK/BASIC, PROC, ASSEMBLY), and the various utility programs.

3. The FILES section contains a minimum set of PICK files, including the SYSTEM dictionary, a SYSPROG account, and the POINTER-FILE, SYS-ERRS, ERRMSG and ACC files. Each account is preceded by a tape label containing the account name, and is followed by an EOF. The last account on the tape is followed by two (2) EOF marks (called an EOD, or END-OF-DATA mark).

A FILE-SAVE tape contains only the third section--Files. There are no coldstart nor ABS sections on FILE-SAVE tapes, only files.

10.25 FILE-RESTORE

File-restores load previously saved files into the system.

A file-restore is initiated from a bootstrap or from the PROC :FILES.

Sequence of Events in File-restores

The first event in a complete file-restore is the initialization of available overflow space to the complete range on the system from the process workspaces (WSSTART) forward to the end of disk (MAXFID).

The file-restore process then proceeds to build the system. It creates the SYSTEM dictionary and clears it. It reads the first account from tape and sets up its master dictionary (MD); a pointer to the MD is placed in the SYSTEM dictionary. The file-restore process next gets the first file for that account; it creates a space for it and places a pointer to it in the account's Master Dictionary. Next is the data file, which is restored in one of two ways:

1. The slow method. The file is created, a pointer is added to the dictionary, and then the data is loaded. Each item must be hashed in order to determine its group. This method is necessary if reallocation is being done, or if the file is the POINTER-FILE.

2. The fast method. The file is created and items are loaded group by group; no hashing is necessary, since the group allocation is not changing. After the file is completely loaded, a pointer is placed in the dictionary. This is the normal method.

The system determines the appropriate method.

After all the data files and items for the first file have been restored, the items in the file dictionary are loaded. The next file's dictionary and data sections are restored in the same manner. When one account is finished, the next account is restored. After all the accounts have been restored, the SYSTEM dictionary is restored. This completes the file-restore.

Account-restores proceed in the same sequence, except that the SYSTEM Dictionary is already present, and only the pointer to the account Master Dictionary is added to it.

Console Listing Accompanying File-restore

The figure below is an example of a file-restore listing. Each line corresponds to a file pointer. Each line is indented in accordance with the level of the file in which the pointer is placed. The file name is first followed by the base, modulo, and separation of the file as it is being restored. An (S) following the line indicates that the pointer has the same base as some other pointer already listed and the file has already been created.

Terminal Response for Additional Reels

If the end-of-tape mark is reached before the system finishes the routine it is executing, the system sends a 'mount next tape' message. When the next tape is mounted, the process waits for the character C to be entered. The tape label on the new tape is compared with the previous label. If the tape label is invalid, a message is displayed and the system waits for the correct tape to be mounted and the character C to be entered.

The 'incorrect tape label' message can be overridden by entering an O at the prompt. The O response causes the system to accept the new reel.

```
SPOOLER STARTED

SYSTEM 8138,1,1                          SYSTEM dictionary pointer
 BLOCK-CONVERT 8138,11,1                 BLOCK-CONVERT
 SYSTEM-ERRORS 8171,11,1                 Pointer to SYSTEM-ERRORS MD
  SYSTEM-ERRORS 8182,11,1                SYSTEM File Dictionary
   SYSTEM-ERRORS 8193,11,1               SYSTEM Data Section
 PROCLIB 8204,2,3,1                      Pointer to PROCLIB MD
 SYSTEM 8138,11,1 (S)                    Pointer back to SYSTEM
 SYSPROG                                 Pointer to SYSPROG MD
  CURSOR 8252,1,1                        DICT of SM file
    .
    .
    .
 ACC 9265,1,1                            SYSTEM pointer to ACC MD
  ACC 9266,1,1                           DICT ACC file in ACC account
   ACC 9267,1,1                          DATA ACC file
```

Sample FILE-RESTORE Console Listing.

10.26 ERROR RECOVERY DURING FILE LOADS

If parity errors or other errors mar the files section of a FILE-SAVE tape, some data may be lost. The file-restore will continue, but operator assistance may be needed.

Parity Error Recovery Procedure

If a parity error is detected on a file restore, the following prompt is displayed:

```
PARITY ERROR!  ENTER A TO TRY AGAIN
I TO IGNORE?
```

To retry, enter A. To accept the data block as it is without data correction, enter I. The specific item and file affected cannot be determined except as can be judged by the tape position and the current set of files which have not been completed.

Recovery From Destroyed Pointers

If tape information identifying a file is destroyed, it may be impossible for the restore to create that file and subsequent files in the right order. The following messaged is displayed:

```
ERROR IN DSEGMENT
< ff.ddd
LEVEL (1-3)?
```

where "ff.ddd" gives the frame and hex displacement of the software location at which the error was detected.

To continue, enter one of the following:

1 - Search for and continue with the next account on tape
2 - Search for the next dictionary file on tape
3 - Search for the next data file on tape

The response requires the operator's judgment as to the positioning of files on the tape and the total situation.

10.27 SELECTIVE RESTORES

SEL-RESTORE is used to selectively restore individual files or items from a system or account file-save tape.

Selective restores are performed as follows:

1. Log on to the account with the file to be restored.

2. Mount the tape. NOTE: Selective-restores may be started from any place on any reel of a multi-tape file-save. To save time in searching a tape, consult the STAT-FILE listing to determine the reel on which the file's data starts and mount that reel.

3. Attach the tape unit (T-ATT).

4. To start the restore, enter:

```
SEL-RESTORE file.name item.list {(options)
```

where

file.name — file in which items are placed; this file must be defined on the account from which the restore is run.

item.list — items eligible for restore; an asterisk (*) may be specified as the item.list to indicate all items on the tape

options — the available options are

- A — tape is positioned in the desired account
- C — This option has effect when the N option is used; it causes every item before the next end of file to be a candidate for restore. This ensures that data can be restored even if a D pointer is damaged on the tape.
- F — display all file names for all accounts; this is not compatible with the N option
- I — item-ids of the restored items are not to be printed
- N — file is to be identified on tape by its file number

  NOTE: the file number can be found on the statistics file print-out for the appropriate file save.
- O — overlay items already on the file.
- S — suppresses 'items on file' message

If the N option is used, the following prompt is displayed:

```
FILE #:
```

Enter the file number.

If the N option is not used, the following prompts are displayed:

```
ACCOUNT NAME ON TAPE:
FILE NAME:
```

The account name is the name of the account under which the file was saved on tape, and file name is the name of the file as it appears on the tape. If <RETURN> is pressed at the file name prompt, the account's Master Dictionary (MD) is restored.

As the tape is searched, the file names on it are printed, along with the file numbers; names are indented one space for account names, two spaces for dictionaries, and three for data file names.

Hints on using SEL-RESTORE

- If a STAT-FILE listing for the tape is available, ensure that the account names and file names are on the tape.

- If in doubt about the contents of the tape, the files can be listed by using a SEL-RESTORE of the form:

```
:SEL-RESTORE TEMP * (F
ACCOUNT-NAME ON TAPE: XXXXX
FILE-NAME: YYYYY
```

  XXXXX and YYYYY are fake names that cause the SEL-RESTORE to search the tape for non-existent data; the F option indicates that file names are printed out as encountered, along with the file numbers.

- In restoring both the dictionary and data section of a file, restore the dictionary first (DICT filename). The dictionary items FOLLOW the data items, so for large files, there may be a considerable pause after the time that the system has found the file (it stops the printout), and the actual restore of the items.

- At any point, the tape may be moved back (T-BCK (n)), or forward-spaced (T-FWD (n)) to position it, and a SEL-RESTORE with the A or N options may be started; this may be faster than restarting the tape from the beginning when restoring both the dictionary and the data sections of a file, or when restoring multiple files.

- Account dictionaries (master dictionary items) FOLLOW ALL OTHER FILES for the account on the tape.

10.28 SYSTEM BACKUP : FILE-SAVE

The PICK system has the ability to save the entire disk data base on magnetic tape and to restore the tape copy, entirely or selectively, to disk. It is this procedure that provides backup in the event of a catastrophic failure or error.
IT IS YOUR RESPONSIBLITY TO DO SAVES FREQUENTLY ENOUGH TO ENSURE ADEQUATE BACKUP FOR YOUR PARTICULAR SITUATION!

The FILE-SAVE procedure protects your valuable data base by creating an off-line copy of it on magnetic tape. Tape is an inexpensive commodity when compared to the time and effort invested in your data base. It is vital that you protect that investment through adequate backup. As a MINIMUM pratice you should have separate daily backup tape-sets for one week's time and a monthly backup for each month in the previous year. Some situations may also need a weekly backup cycle for the past month. That is, use a separate tape-set for each day of the week, one for each week of the month and one for each month of the year. The longer cycle tape-sets should be stored off premises to provide protection in the event of physical damage such as fire.

ONLY YOU CAN DETERMINE WHAT IS ADEQUATE FOR THE PROTECTION OF YOUR DATA!

FILE-SAVEs are performed as follows:

1. Mount the tape reel onto which you intend to save your data.

2. Enter

   FILE-SAVE

3. Several tape functions are performed automatically, then the following prompts are displayed:

   List files saved to Crt or Printer? (C or P) =

   The FILE-SAVE procedure normally creates a list on the terminal of the files it finds as it saves the data base. It outputs error messages if it encounters unusual or illegal conditions, but it attempts to continue to save data. To send the listing to the printer, enter Y.

   Send STAT-FILE report to printer? (Y or N) =

   The FILE-SAVE generates a statistics report of the saved data. To print the report, enter Y.

   T-DUMP STAT-FILE to tape at end of FILE-SAVE? (Y or N) =

   To save the report on to the file-save tape, enter Y.


```
Verify save with dummy SEL-RESTORE?          (Y or N) =
```

The verify checks for parity errors on the save media.

```
Enter tape label text (without embedded spaces) or <CR> for none
   Tape Label =
```

4. FILE-SAVE then saves your data.

5. Operator intervention is required only if the data to be saved exceeds one tape reel.

You now have a complete backup of your disk data base.

10.29 THE SAVE VERB

SAVE is the verb that performs a FILE-SAVE; it is called by the FILE-SAVE PROC.

The FILE-SAVE PROC sets up a sentence using the SAVE verb.

FORMAT:

SAVE {(options)}

OPTION	MEANING
D	Data area is saved. This option must be present if any files are to be saved.
F	File names are printed. If (F) is not specified, just the SYSTEM file and account-names are listed.
G	Group Format Errors are repaired. GFEs are also logged in the STAT-FILE, if the (S) option is present.
I	Account save.
P	Output (list of file names) goes to the line printer. If (P) is not specified, all output goes to the user's terminal.
S	STAT-FILE items are stored, one for each file saved. Must be present if a STAT-FILE listing is to be made after the FILE-SAVE.
T	Output to Magnetic Tape. If the (T) option is not specified, nothing is be written on magnetic tape. However, the STAT-FILE will be generated if the (S) option is used.

Files whose file definition items have a "DX" in line 1 are not saved. Thus, any data file, dictionary or even an entire account may be prevented from taking up space on the FILE-SAVE tape.

Files whose file definition items have a "DY" in line 1 are saved, but none of the items in the file or sub-files will be saved. The data section of the STAT-FILE, for instance, has a "DY" code, because the data is not valid after a file-restore, and needs not be saved.

To prevent spurious Group Format Error messages from occurring on other lines while the FILE-SAVE is running, the SAVE processor locks groups as it saves them. Up to 4 groups may be locked at one time by a file-save process. These groups are those containing the following:

1. The SYSTEM dictionary pointer for the account being saved.
2. The file dictionary pointer for the dictionary of the file being saved. This would be a group in the account's MD.
3. The group in the data file being saved.

4. A group in the dictionary of the ACC file.

If a process on another line tries to access data in a locked group, that process is paused until the file-save finishes saving all the items in that group and unlocks it.

If the (T) option is specified, the SAVE processor will prompt the user's terminal:

FILE-SAVE TAPE LABEL =

The response is written on the tape as part of the tape label.

10.29.1 MULTIPLE REEL SAVES

When the data to be saved exceeds the capacity of the mounted reel, a MOUNT NEXT REEL message appears on the terminal screen. A pound sign (#) prompt follows the message.

Remove the tape reel, which should have rewound itself. Mount and position the next reel to the BOT (Beginning Of Tape) mark. Make sure that the media is write-enabled and that the tape drive is on-line. Now enter one of the following characters at the #, as appropriate:

C - CONTINUE
O - OVERWRITE (used in cases of erroneous tape labels)
Q - QUIT

This procedure also holds true for RESTORING multiple reels.

10.12 THE ACCOUNTING HISTORY FILE: SUMMARY AND EXAMPLES

This topic summarizes the formats of the active user items and the accounting history items in the Accounting History File. Also presented are sample entries for the Accounting History File.

The first figure summarizes the attributes for the active user items and the accounting history items. The second figure shows a sample sorted listing of the active users (users with a value for attribute A1) via an ACCESS SORT statement. The third figure shows a sample listing of the accounting history item for user SMITH via an ACCESS LIST statement.

ATTRIBUTE NUMBER	'ACC' DICTIONARY NAME	ACTIVE USER ITEM	ACCOUNTING HISTORY ITEM
	(item-id)	Four-character hexadecimal PCB-FID	User name#lineno
1	NAME	User name	Not used
2	DATE	Date logged on	Not used
3	TIME	Time logged on	Not used
4	DATES		Dates logged on
5	TIMES		Times logged on
6	CONN		Connect time
7	UNITS		Charge-units
8	PAGES		Number of printer pages generated.

Summary of Active User Items and Accounting History Items

```
>LISTU [CR]

CH# PCBF NAME..... TIME... DATE.... LOCATION.....

 00 0200 CM           11:02AM 03/22/78 Channel 0
 01 0220 SYSPROG      12:10PM 03/22/78 Channel 1
 02 0240 EL-ROD       09:11AM 03/22/78 Channel 2
 03 0260 LC           06:59AM 03/22/78 Channel 3
 05 02A0 HVE          09:55AM 03/22/78 Channel 5
*06 02C0 CM           11:25AM 03/22/78 Channel 6
 07 02E0 BUGEYE       01:29PM 03/21/78 Channel 7
 10 0340 JT           11:34AM 03/22/78 Channel 10
```

Sample sorted listing of active user items (using LISTU).

```
>LIST ACC = "SMITH]"              (selects items with item-ids
                                   starting with the string "SMITH")
PAGE   1                                    12:17:22  22 MAR 1978

ACC..... DATE. TIME... CONN... UNITS.. PAGES
               *       *       *       *
SMITH#0  01/13   16:56   00:04       9
         01/14   10:13   00:00       5
                 10:15   00:01     343
         02/06   17:02   00:18      41
         02/09   10:21   00:17     690
         02/23   07:58   00:01      27
         03/09   11:35   01:57     378
                 16:05   00:22      94
SMITH#5  01/13   12:48   02:25     160      5
                 15:20   00:05      14
                 15:25   00:00       2
                 15:28   00:17     110
                 16:20   02:55    2575     16
                 19:15   00:00      13
         01/16   09:41   06:13    1853      6
                 15:55   00:12      15

2 ITEMS LISTED.
```

Sample listing of accounting-history item for user "SMITH".

10.13 THE ACCOUNTING HISTORY FILE: PERIODIC CLEARING

To avoid overflowing the accounting history items in the Accounting History file for a specific user, the items should be periodically cleared.

To clear the accounting history items from the ACC file, follow the steps detailed in the first figure.

The point of overflow is determined by the activity of the user-account (however, approximately 1000 Logon/Logoffs are allowed). This point can be calculated by following the procedure detailed in the second figure.

If the accounting history item for a user-account does exceed the available workspace, the user will be logged off, but the Accounting History File will not be updated. To recover from this situation, follow the procedure detailed below.

1. Logon to the SYSPROG account.

2. Type the following (if you need a listing only):

```
>SORT ACC WITH NAME LPTR [CR]
```

3. Type the following:

```
>SELECT ACC WITH NAME [CR]
>DELETE ACC  [CR]
```

Procedure to Clear all Accounting History Items.

1. Use the WHAT verb to determine the number of additional work-space frames allocated for the system (parameter WSSIZE in the WHAT display). Multiply this figure by 500 and add 3000.

2. To determine the current size, type:

```
>STAT ACC ACC-SIZE 'user-name'  [CR]
```

This will produce the following output:

```
STATISTICS OF ACC-SIZE:
TOTAL = xxx AVERAGE = yyy COUNT = zzz
```

3. If the value displayed for TOTAL in step 2 (i.e., xxx) approaches the value calculated in step 1, then the user-account is approaching the overflow point.

Determining the point of overflow for an accounting-history item.

10.14 FILE STRUCTURE: THE ITEM AND GROUP COMMANDS

The ITEM and GROUP commands provide information about the item and group structure of Pick files. Output can be displayed at the terminal or optionally directed to the line printer.

FORMAT:

ITEM file-name item-id {(options)}

This command displays the base FID of the group into which the specified item-id hashes. If the item is not already on file, the message "ITEM NOT FOUND" is displayed. In addition, every item-id in that group is listed along with a character count of the item (in hex). At the end of the list the following message is displayed:

n ITEMS m BYTES p/q FRAMES

where:

n is the number of items in the group.
m is the total number of bytes used in the group.
p is the number of full frames in the group.
q is the number of bytes used in the last frame of the group.

Valid options for this command are as follows:

P - Direct output to line printer.
S - Suppress item list.

FORMAT:

GROUP file-name {(options)}

This command displays the base FID of each group in the specified file. In addition, every item-id in the group is listed along with a character count of the item (in hex). At the end of the list for each group the following message is displayed:

n ITEMS m BYTES p/q FRAMES

where:

n is the number of items in the group.
m is the total number bytes used in the group.
p is the number of full frames in the group.
q is the number of bytes used in the last frame of the group.

Valid options for this command are as follows:

P - Direct output to line printer.
S - Suppress item list.

```
>ITEM M/DICT A [CR]

27121
0022 FILE-DOC
001C bd
0009 A
0011 T-ATT
000F DUMP
0018 B/ADD
000F DIVX
0014 EDIT-LIST
0028 V/CONV
0022 LISTU
0019 V/MIN
001B ACCOUNT-RESTORE
001D D/CODE
0028 SL
0023 INST-INDEX
0047 SAL
0072 TB
000E SAVE
18 ITEMS 591 BYTES 1/91 FRAMES
```

Displaying data in a group using the ITEM command.

10.15 FILE STRUCTURE: THE ISTAT AND HASH-TEST COMMANDS

ISTAT and HASH-TEST are ACCESS verbs that produce file hashing histograms, ISTAT for specified file items and HASH-TEST on the basis of a user-specified test modulo.

ISTAT

An ACCESS sentence using the ISTAT command is constructed as illustrated below. The ISTAT command provides a file hashing histogram for the selected items in the selected file, as illustrated by the examples. For further information regarding file hashing, refer to the section of this manual titled VIRTUAL MEMORY OPERATING SYSTEM.

HASH-TEST

HASH-TEST produces a file hashing histogram as a result of a user-specified test modulo. The general form of this verb is as follows:

```
HASH-TEST (DICT) file-name (item-list) (selection-criteria)
```

```
>ISTAT PARCEL [CR]

FILE= PARCEL MODULO= 3 SEPAR= 1                13:50:42  22 MAR 1978
FRAMES BYTES ITMS
000002 00757 027 *>>>>>>>>>>>>>>>>>>>>>>>>>>
000002 00836 030 *>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
000002 00785 028 *>>>>>>>>>>>>>>>>>>>>>>>>>>

ITEM COUNT=          85, BYTE COUNT=      2378, AVG. BYTES/ITEM=     27.9
AVG. ITEMS/GROUP=  28.3, STD. DEVIATION=   1.5, AVG. BYTES/GROUP=   792.6.
```

Sample usage of the ISTAT command.

```
>HASH-TEST PARCEL [CR]

TEST MODULO: 9 [CR]
FILE= PARCEL MODULO= 9 SEPAR= 1                13:50:55  22 MAR 1978
FRAMES BYTES ITMS
000001 00256 009 *>>>>>>>>>
000001 00281 010 *>>>>>>>>>>
000001 00255 009 *>>>>>>>>>
000001 00229 008 *>>>>>>>>
000001 00248 009 *>>>>>>>>>
000001 00251 009 *>>>>>>>>>
000001 00272 010 *>>>>>>>>>>
000001 00307 011 *>>>>>>>>>>>
000001 00279 010 *>>>>>>>>>>

ITEM COUNT=          85, BYTE COUNT=      2378, AVG. BYTES/ITEM=     27.9
AVG. ITEMS/GROUP=   9.4, STD. DEVIATION=    .8, AVG. BYTES/GROUP=   264.2.
```

Sample usage of the HASH-TEST verb.

10.16 DETERMINING NATURE OF GROUP FORMAT ERRORS

10.16.1 GROUP DEFINITION

The term group is used to specify one 'bucket' of storage. A file is made up of a collection of groups, such that there are the same number of groups as the number specified for the modulo of the file. Put another way, the modulo of the file specifies the number of groups which make up the file.

The hashing algorithm takes the specified item-id and decides in which group it is or should be stored. The file retrieval or storage routine then searches that group for the specified item. The hashing algorithm may be thought of as dividing the item-id by the modulo in order to obtain the remainder. This remainder is then the 'group number', and specifies the group which is to be searched.

Within each group the items are stored physically end to end. Each item is made up of a count field, a key, and the data. The documentation for this system has conventionally used the term 'item-id' in place of the term 'key'. It remains that the item-id is the key which is used to look up the location of the item.

The count field exists only in a file representation of the item. It is a sixteen-bit binary number, such that the high-order bit is zero, represented in the file in ASCII hexadecimal notation, and as such takes up four bytes of storage. It immediately precedes the item-id in the file. If the item in question is the first item in the group, the count field starts in the first data byte in the frame. If the item is not the first item in the group, then the count field starts at the first byte after the termination mark of the last item.

The count field is used as a pointer to the end of the item. The end of the item must be an attribute mark followed by a segment mark. If the count field does not point to this pattern, there is a group format error, and the group format error handler will be entered.

10.16.2 GROUP FORMAT ERRORS A GROUP FORMAT ERROR IS THE RESULT OF A HARDWARE ERROR!

A group format error is sensed if the count field does not point at an attribute mark, segment mark sequence. This may occur if the count is wrong, or if the data at the end of the item is wrong.

The count field is definitely wrong if any of the four digits which make up the count field are not ASCII hexadecimal digits, which are X'30' - X'39' or X'41' - X'46', which are 0-9 and A-F.

The end of item data may be wrong if the count field contains the wrong ASCII hexadecimal digits, or if the end of item data is actually wrong.

The end of item data may be wrong in several ways. If the item is contained in a frame, then the end of item data may be wrong in the ways that the the count field may be wrong. If the item spans a frame boundary, certain other mechanisms come into play. If a process was in the process of updating an item, to the extent that the first frame containing the item was written to disk, but that the last frame was not written when the process was interrupted by something like a cold start, then a group format error will occur. If the overflow handler becomes confused, the frames attached to a group may be acquired by another data file or by a print file. The difference should be obvious on inspection, using the DUMP verb. Print files do not normally contain attribute or value marks and data files do not normally contain carraige-return, line-feed sequences.

If the damaged frame is the result of an incomplete update, then the difficulty is localized. Repair of this group will usually attend to the matter. If the damage appears to be due to co-ownership of the frame, the problem may be greater. In this case it is best to leave the frame with the frame to which it has a back-link, presuming that the data is consistent in that chain. Then cut the forward link in the spurious chain and terminate the group.

The effect of the group format error handler is to terminate the group at the end of the last consistent item and cut the forward link out of the last acceptable frame in the group. The rest of the overflow is intentionally lost, because of the effect of having two copies of the same frame referenced in the overflow chain.

The one case in which the group will not be terminated is when a print file has meandered across the base of the file. In this case it is probably best to recreate the file and selectively restore it. The old file pointer should be thrown away. Do not use the DELETE-FILE verb on the old file, because this will further muddy the condition of the overflow handler.

10.16.3 RECOVERY FROM GFE's

If a group format error is encountered, the system will invoke the group format error handler. This processor will print the error message to the terminal and wait for an operator response. The valid operator responses are:

'D' - which will enter the system debugger.

'E' - which will end the process and return to TCL.

'F' - which will allow the GFE handler to fix the error and continue.

NOTE that fixing the error will undoubtedly cause the loss of at least one data item. This record normally must be manually recovered! The recovery strategy is to identify the file affected and do a SEL- RESTORE on the file. It is best to do this as soon after the group format error is noticed as possible.

10.17 GENERATING CHECKSUMS: THE CHECK-SUM COMMAND

The CHECK-SUM command generates a checksum for file items, thus providing a means to determine if data in a file has been changed.

FORMAT:

```
CHECK-SUM {DICT} file-name {item-list} {attribute} {selection-criteria}
```

A checksum is generated for items in the specified file, or subset of items if the optional "item-list" and/or "selection-criteria" appear. Furthermore, the checksum may be calculated for one specified attribute. If no attribute is specified, the 1st default attribute will be used. If there is no default attribute, or if the AMC is 9999, the entire item will be included. The checksum will include the binary value of each character times a positional value. This yields a checksum which has a high probability of being unique for a given character string. The dictionary portion is checksummed if the "DICT" option appears. (A checksum is the arithmetic total, disregarding overflow, of all bytes in the selected items.)

A message is output, giving checksum statistics, in the following form:

```
        BYTE STATISTICS FOR  file-name (or attribute name):
   TOTAL = t AVERAGE = a ITEMS = i CKSUM= c BITS = b
```

where:

t is the total number of bytes in the attribute (or item) included
a is the average number of bytes
i is the number of items
c is the checksum
b is a bit count

The attribute mark trailing the specified attribute (or item) will be included in the statistics.

To use checksums, the user should issue CHECK-SUM commands for all files, or portions of files, to be verified and keep the output statistics. Subsequently, the CHECK-SUM commands can be reissued to verify that the checksum statistics have not changed. The checksum must be recalculated whenever the user updates the file!

10.18 SYSTEM PROGRAMMER (SYSPROG) ACCOUNT

Several special facilities are normally used from the System Programmer (SYSPROG) Account. Procedures on this account are normally performed by persons more familiar with the overall operation of the system.

To log on to the SYSPROG Account, type the following:

LOGON PLEASE: SYSPROG,password [CR]

where "password" is the appropriate password set up for SYSPROG. Alternate logon names (such as SP) may be used.

CREATE-ACCOUNT	DELETE-ACCOUNT
ACCOUNT-RESTORE	SAVE
BUFFERS	SEL-RESTORE
LOCK-FRAME	UNLOCK-FRAME
:FILES	:ABSLOAD
:ABS/FILES	WHAT

Some SYSPROG Verbs and Procs.

10.19 AVAILABLE SYSTEM SPACE: THE POVF COMMAND

The POVF verb displays the system overflow table.

FORMAT:

POVF {(P)

The POVF verb displays the contents of the system overflow table.

The P option forces all printed output to the line printer. the first line of output is the FID of the first frame in linked overflow, folowed by the number of frames in the linked chain. the next lines (up to 16) describe blocks of contiguous overflow, and have the following format:

m - n : p m - n : p

where:

m is the first frame of a contiguous block.
n is the last frame of the block.
p is the number of frames in the block.

The total number of frames contained in all the contiguous overflow is then printed (using error message number 293):

TOTAL NUMBER OF CONTIGUOUS FRAMES :number

10.20 CREATING ACCOUNTS

The CREATE-ACCOUNT PROC is used to create new user-accounts.

CREATE-ACCOUNT PROC

The CREATE-ACCOUNT PROC generates a new account according to given specifications. It then copies the contents of the NEWAC file (the prototype M/DICT) to the new user M/DICT. Finally, it adds a file synonym (Q item) to the account into SYSPROG's M/DICT. The CREATE-ACCOUNT PROC is invoked by typing in the PROC name:

```
>CREATE-ACCOUNT  [CR]
```

The PROC then prompts the user for the required information, as shown below.

NOTE: The CREATE-ACCOUNT PROC should not be used to create a new synonym to an existing account; this should be done by using the EDITOR to create the file synonym definition item (Q-item) in the SYSTEM dictionary.

```
>CREATE-ACCOUNT          PROC is typed in at TCL.
ACCOUNT NAME?SHERRY      Anything but [CR] is legal.
L/RET-CODE(S)?AAA]BBB    Multi-valued retrieval code.
L/UPD-CODE(S)?           [CR] means no lock code.
PRIVILEGES?              [CR] means SYS0. May be SYS0, SYS1, or SYS2.
MOD, SEP?37,1            [CR] defaults to 29,1.

CREATE-FILE (DICT SHERRY 37,1

[417] FILE 'SHERRY' CREATED; BASE= 34593 MODULO= 37 SEPAR = 1.

280 ITEMS COPIED
'SHERRY' ADDED
'SHERRY' UPDATED
PASSWORD?R2D2            User's LOGON password.

FINISHED
```

Sample CREATE-ACCOUNT Usage.

10.21 DELETE-ACCOUNT

DELETE-ACCOUNT deletes an account and all its files from an PICK system.

DELETE-ACCOUNT is a PROC which runs the program DEL-ACC. The program lists all the files in the specified account, then requests verification to delete the account. The files may be listed on the terminal or the printer.

Requirements to run DELETE-ACCOUNT:

1. You must be logged on to SYSPROG.
2. SYSPROG must have Q-pointers to the MD of the account, and to SYSTEM.
3. D-items must exist in DICT SYSTEM for SYSPROG and the account name.
4. SYSPROG must have access to SYSTEM and all files on the account to be deleted.

ALL USERS SHOULD LOG OFF before running this because an item in the SYSTEM dictionary will be deleted.

```
>DELETE-ACCOUNT                    PROC name

Account Name ?SHERRY               Enter account name

List Files on Printer (Y/N) ?      To list files on printer, enter Y.

Files to be Deleted in Account:  SHERRY  11:29:14  02 APR 88  PAGE   1

FILE      Type      BASE    MOD  SEP

GEN/LED   D         85344   1    1
GEN/LED   D         49911   231  1
BP        D         44319   17   5

Still want to Delete Account SHERRY? Y
                                   To delete the account, enter Y.
```

Sample DELETE-ACCOUNT usage.

10.22 FILE STATISTICS REPORT

The File Statistics Report provides a list of file parameters, such as name, base, modulo, and file size. It also provides the order of files on a FILE-SAVE tape. The report is automatically generated by running a FILE-SAVE, or may be generated at any time by using the PROC LIST-FILE-STATS.

The report is broken down by account, with a line of information generated for each file in the account that includes:

item.id
name
base, modulo, and separation
total file size
total number of frames used
utilization of file space
number of Group Format Errors (GFEs)

If the report is being sent to a printer that prints 132 columns, the following additional information is included:

average item size
average number of items per group
pad space; that is, unused space

A total line is generated for each account.

The information for the report is kept in the STAT-FILE on the account that does the FILE-SAVE; this is usually the SYSPROG account. The FILE-SAVE process creates one item in the STAT-FILE for each file saved on the file-save tape. The item-ids in the STAT-FILE are of the form

t:n

where

t tape reel number where the file was dumped (this is 0 if the SAVE was run without dumping data to the tape)

n file number; this file-number is used in the selective restoration of files using SEL-RESTORE.

The NAME field of the items in the STAT-FILE contains data in the form:

dictname (dictionary file)
dictname*dataname (data file)

When a FILE-SAVE is started, the STAT-FILE data area is cleared and the current file statistics information is written into the data area. The STAT-FILE data area is also empty after a file-restore is done, because attribute 1 of the file definition is a DY. This is desirable as the statistics are no longer applicable.

```
ITM-ID R#  ID NAME..... BASE. MOD S ...SIZE FRAMES %UT* GFE

1:367  1  367 ACC          9253   1 1      29      1   5
1:368  1  368 ACC          9254   1 1   2,191      5  87
1:369  1  369 ACC*ACC       9255  13 1      54     13   0
                                        ------    --- ---   ---
*** Totals for user:  ACC                2,274     19  23

1:2    1    2 BLOCK-CONVERT  8149  11 1   8,324     24  69
     .
     .

*The utilization of file space is derived by dividing the size  of the
 file by the number of frames.
```

10.23 UTILITY VERBS: STRIP-SOURCE, LOCK-FRAME, UNLOCK-FRAME, CHARGES, AND CHARGE-TO

This topic describes a number of special utility verbs.

STRIP-SOURCE Verb

The STRIP-SOURCE verb is a TCL-II verb used to remove the source code from Assembly Language programs. This frees large amounts of disk space back to the available space pool. Modes with source stripped out can still be verified against the ABS.

FORMAT:

STRIP-SOURCE file-name item-list

After the verb has been invoked, the user is prompted with:

DESTINATION FILE:

The file-name where the stripped object code is to be stored should then be entered. For example:

```
>STRIP-SOURCE PROG * [CR]
DESTINATION FILE-SPROG [CR]
```

Here the file PROG containing source programs is stripped and copied to the file SPROG.

The first six lines of the source item will be copied without source code stripping. Standard Pick Systems convention for source modes has the "FRAME" statement in line 1, and other descriptive information in lines 2 through 6; this information is maintained through the STRIP-SOURCE process.

LOCK-FRAME Verb

The LOCK-FRAME verb may be used to core lock a frame.

FORMAT:

LOCK-FRAME number

where "number" is a decimal frame number. The LOCK-FRAME verb responds with the absolute hexadecimal work address of the memory buffer in which the frame is corelocked. The frame remains corelocked until it is released by the UNLOCK-FRAME verb, or the system is re-booted.

UNLOCK-FRAME Verb

The UNLOCK-FRAME verb clears the corelocked buffer status of the frame indicated.

FORMAT:

UNLOCK-FRAME number

where "number" is a decimal frame number.

CHARGES Verb

The CHARGES verb prints the current computer usage since logon as connect time in minutes and CPU usage in charge-units.

FORMAT:

CHARGES

CHARGE-TO Verb

The CHARGE-TO verb is used to keep track of computer usage for several projects associated with the same logon name.

FORMAT:

CHARGE-TO name

This verb performs the following:

1. Terminates the current charge session by updating the ACC file with the user's accumulated charge-units, line printer pages and connect-time statistics.

2. Changes the logon name to the original name concatenated with an asterisk and then the name following "CHARGE-TO".

For example, if the user is currently logged on to SYSPROG, and he types in the following:

```
>CHARGE-TO PROJECT1 [CR]
```

the LOGON name in the ACC file for the process will be changed to "SYSPROG*PROJECT1".

10.24 SYS-GEN AND FILE-SAVE TAPES: FORMAT

System restore is the process of "bringing up", or creating, the PICK system. A bootstrap program, all system software, and all files can be loaded from magnetic tape. The system configuration is set up at cold-start time.

A system can be restored from a SYS-GEN tape or a file-save tape. There are three sections on a SYS-GEN tape:

1. The bootstrap section contains the MONITOR, the configurator, and some virtual program frames needed to bootstrap the system. There are 33 tape records in this section, followed by an END-OF-FILE mark (EOF).

2. The ABS section, which contains the system software. This section is preceded by a tape label, which contains the release level, and terminated by an EOF. This software makes up the PICK Operating System, the Language processors (ACCESS, PICK/BASIC, PROC, ASSEMBLY), and the various utility programs.

3. The FILES section contains a minimum set of PICK files, including the SYSTEM dictionary, a SYSPROG account, and the POINTER-FILE, SYS-ERRS, ERRMSG and ACC files. Each account is preceded by a tape label containing the account name, and is followed by an EOF. The last account on the tape is followed by two (2) EOF marks (called an EOD, or END-OF-DATA mark).

A FILE-SAVE tape contains only the third section--Files. There are no coldstart nor ABS sections on FILE-SAVE tapes, only files.

10.25 FILE-RESTORE

File-restores load previously saved files into the system.

A file-restore is initiated from a bootstrap or from the PROC :FILES.

Sequence of Events in File-restores

The first event in a complete file-restore is the initialization of available overflow space to the complete range on the system from the process workspaces (WSSTART) forward to the end of disk (MAXFID).

The file-restore process then proceeds to build the system. It creates the SYSTEM dictionary and clears it. It reads the first account from tape and sets up its master dictionary (MD); a pointer to the MD is placed in the SYSTEM dictionary. The file-restore process next gets the first file for that account; it creates a space for it and places a pointer to it in the account's Master Dictionary. Next is the data file, which is restored in one of two ways:

1. The slow method. The file is created, a pointer is added to the dictionary, and then the data is loaded. Each item must be hashed in order to determine its group. This method is necessary if reallocation is being done, or if the file is the POINTER-FILE.

2. The fast method. The file is created and items are loaded group by group; no hashing is necessary, since the group allocation is not changing. After the file is completely loaded, a pointer is placed in the dictionary. This is the normal method.

The system determines the appropriate method.

After all the data files and items for the first file have been restored, the items in the file dictionary are loaded. The next file's dictionary and data sections are restored in the same manner. When one account is finished, the next account is restored. After all the accounts have been restored, the SYSTEM dictionary is restored. This completes the file-restore.

Account-restores proceed in the same sequence, except that the SYSTEM Dictionary is already present, and only the pointer to the account Master Dictionary is added to it.

Console Listing Accompanying File-restore

The figure below is an example of a file-restore listing. Each line corresponds to a file pointer. Each line is indented in accordance with the level of the file in which the pointer is placed. The file name is first followed by the base, modulo, and separation of the file as it is being restored. An (S) following the line indicates that the pointer has the same base as some other pointer already listed and the file has already been created.

Terminal Response for Additional Reels

If the end-of-tape mark is reached before the system finishes the routine it is executing, the system sends a 'mount next tape' message. When the next tape is mounted, the process waits for the character C to be entered. The tape label on the new tape is compared with the previous label. If the tape label is invalid, a message is displayed and the system waits for the correct tape to be mounted and the character C to be entered.

The 'incorrect tape label' message can be overridden by entering an O at the prompt. The O response causes the system to accept the new reel.

```
SPOOLER STARTED

SYSTEM 8138,1,1                          SYSTEM dictionary pointer
 BLOCK-CONVERT 8138,11,1                 BLOCK-CONVERT
 SYSTEM-ERRORS 8171,11,1                 Pointer to SYSTEM-ERRORS MD
  SYSTEM-ERRORS 8182,11,1                SYSTEM File Dictionary
   SYSTEM-ERRORS 8193,11,1               SYSTEM Data Section
 PROCLIB 8204,2,3,1                      Pointer to PROCLIB MD
 SYSTEM 8138,11,1 (S)                    Pointer back to SYSTEM
 SYSPROG                                 Pointer to SYSPROG MD
  CURSOR 8252,1,1                        DICT of SM file
   .
   .
   .
 ACC 9265,1,1                            SYSTEM pointer to ACC MD
  ACC 9266,1,1                           DICT ACC file in ACC account
   ACC 9267,1,1                          DATA ACC file
```

Sample FILE-RESTORE Console Listing.

10.26 ERROR RECOVERY DURING FILE LOADS

If parity errors or other errors mar the files section of a FILE-SAVE tape, some data may be lost. The file-restore will continue, but operator assistance may be needed.

Parity Error Recovery Procedure

If a parity error is detected on a file restore, the following prompt is displayed:

```
PARITY ERROR!  ENTER A TO TRY AGAIN
I TO IGNORE?
```

To retry, enter A. To accept the data block as it is without data correction, enter I. The specific item and file affected cannot be determined except as can be judged by the tape position and the current set of files which have not been completed.

Recovery From Destroyed Pointers

If tape information identifying a file is destroyed, it may be impossible for the restore to create that file and subsequent files in the right order. The following messaged is displayed:

```
ERROR IN DSEGMENT
< ff.ddd
LEVEL (1-3)?
```

where "ff.ddd" gives the frame and hex displacement of the software location at which the error was detected.

To continue, enter one of the following:

1 - Search for and continue with the next account on tape
2 - Search for the next dictionary file on tape
3 - Search for the next data file on tape

The response requires the operator's judgment as to the positioning of files on the tape and the total situation.

10.27 SELECTIVE RESTORES

SEL-RESTORE is used to selectively restore individual files or items from a system or account file-save tape.

Selective restores are performed as follows:

1. Log on to the account with the file to be restored.

2. Mount the tape. NOTE: Selective-restores may be started from any place on any reel of a multi-tape file-save. To save time in searching a tape, consult the STAT-FILE listing to determine the reel on which the file's data starts and mount that reel.

3. Attach the tape unit (T-ATT).

4. To start the restore, enter:

```
SEL-RESTORE file.name item.list {(options}
```

where

file.name — file in which items are placed; this file must be defined on the account from which the restore is run.

item.list — items eligible for restore; an asterisk (*) may be specified as the item.list to indicate all items on the tape

options — the available options are

- A — tape is positioned in the desired account
- C — This option has effect when the N option is used; it causes every item before the next end of file to be a candidate for restore. This ensures that data can be restored even if a D pointer is damaged on the tape.
- F — display all file names for all accounts; this is not compatible with the N option
- I — item-ids of the restored items are not to be printed
- N — file is to be identified on tape by its file number

  NOTE: the file number can be found on the statistics file print-out for the appropriate file save.
- O — overlay items already on the file.
- S — suppresses 'items on file' message

If the N option is used, the following prompt is displayed:

```
FILE #:
```

Enter the file number.

If the N option is not used, the following prompts are displayed:

```
ACCOUNT NAME ON TAPE:
FILE NAME:
```

The account name is the name of the account under which the file was saved on tape, and file name is the name of the file as it appears on the tape. If <RETURN> is pressed at the file name prompt, the account's Master Dictionary (MD) is restored.

As the tape is searched, the file names on it are printed, along with the file numbers; names are indented one space for account names, two spaces for dictionaries, and three for data file names.

Hints on using SEL-RESTORE

- If a STAT-FILE listing for the tape is available, ensure that the account names and file names are on the tape.

- If in doubt about the contents of the tape, the files can be listed by using a SEL-RESTORE of the form:

```
:SEL-RESTORE TEMP * (F
ACCOUNT-NAME ON TAPE: XXXXX
FILE-NAME: YYYYY
```

  XXXXX and YYYYY are fake names that cause the SEL-RESTORE to search the tape for non-existent data; the F option indicates that file names are printed out as encountered, along with the file numbers.

- In restoring both the dictionary and data section of a file, restore the dictionary first (DICT filename). The dictionary items FOLLOW the data items, so for large files, there may be a considerable pause after the time that the system has found the file (it stops the printout), and the actual restore of the items.

- At any point, the tape may be moved back (T-BCK (n)), or forward-spaced (T-FWD (n)) to position it, and a SEL-RESTORE with the A or N options may be started; this may be faster than restarting the tape from the beginning when restoring both the dictionary and the data sections of a file, or when restoring multiple files.

- Account dictionaries (master dictionary items) FOLLOW ALL OTHER FILES for the account on the tape.

10.28 SYSTEM BACKUP : FILE-SAVE

The PICK system has the ability to save the entire disk data base on magnetic tape and to restore the tape copy, entirely or selectively, to disk. It is this procedure that provides backup in the event of a catastrophic failure or error.

IT IS YOUR RESPONSIBLITY TO DO SAVES FREQUENTLY ENOUGH TO ENSURE ADEQUATE BACKUP FOR YOUR PARTICULAR SITUATION!

The FILE-SAVE procedure protects your valuable data base by creating an off-line copy of it on magnetic tape. Tape is an inexpensive commodity when compared to the time and effort invested in your data base. It is vital that you protect that investment through adequate backup. As a MINIMUM pratice you should have separate daily backup tape-sets for one week's time and a monthly backup for each month in the previous year. Some situations may also need a weekly backup cycle for the past month. That is, use a separate tape-set for each day of the week, one for each week of the month and one for each month of the year. The longer cycle tape-sets should be stored off premises to provide protection in the event of physical damage such as fire.

ONLY YOU CAN DETERMINE WHAT IS ADEQUATE FOR THE PROTECTION OF YOUR DATA!

FILE-SAVEs are performed as follows:

1. Mount the tape reel onto which you intend to save your data.

2. Enter

   FILE-SAVE

3. Several tape functions are performed automatically, then the following prompts are displayed:

   List files saved to Crt or Printer? (C or P) -

   The FILE-SAVE procedure normally creates a list on the terminal of the files it finds as it saves the data base. It outputs error messages if it encounters unusual or illegal conditions, but it attempts to continue to save data. To send the listing to the printer, enter Y.

   Send STAT-FILE report to printer? (Y or N) -

   The FILE-SAVE generates a statistics report of the saved data. To print the report, enter Y.

   T-DUMP STAT-FILE to tape at end of FILE-SAVE? (Y or N) -

   To save the report on to the file-save tape, enter Y.


```
Verify save with dummy SEL-RESTORE?          (Y or N) -
```

The verify checks for parity errors on the save media.

```
Enter tape label text (without embedded spaces) or <CR> for none
   Tape Label -
```

4. FILE-SAVE then saves your data.

5. Operator intervention is required only if the data to be saved exceeds one tape reel.

You now have a complete backup of your disk data base.

10.29 THE SAVE VERB

SAVE is the verb that performs a FILE-SAVE; it is called by the FILE-SAVE PROC.

The FILE-SAVE PROC sets up a sentence using the SAVE verb.

FORMAT:

SAVE {(options)}

OPTION	MEANING
D	Data area is saved. This option must be present if any files are to be saved.
F	File names are printed. If (F) is not specified, just the SYSTEM file and account-names are listed.
G	Group Format Errors are repaired. GFEs are also logged in the STAT-FILE, if the (S) option is present.
I	Account save.
P	Output (list of file names) goes to the line printer. If (P) is not specified, all output goes to the user's terminal.
S	STAT-FILE items are stored, one for each file saved. Must be present if a STAT-FILE listing is to be made after the FILE-SAVE.
T	Output to Magnetic Tape. If the (T) option is not specified, nothing is be written on magnetic tape. However, the STAT-FILE will be generated if the (S) option is used.

Files whose file definition items have a "DX" in line 1 are not saved. Thus, any data file, dictionary or even an entire account may be prevented from taking up space on the FILE-SAVE tape.

Files whose file definition items have a "DY" in line 1 are saved, but none of the items in the file or sub-files will be saved. The data section of the STAT-FILE, for instance, has a "DY" code, because the data is not valid after a file-restore, and needs not be saved.

To prevent spurious Group Format Error messages from occurring on other lines while the FILE-SAVE is running, the SAVE processor locks groups as it saves them. Up to 4 groups may be locked at one time by a file-save process. These groups are those containing the following:

1. The SYSTEM dictionary pointer for the account being saved.
2. The file dictionary pointer for the dictionary of the file being saved. This would be a group in the account's MD.
3. The group in the data file being saved.

4. A group in the dictionary of the ACC file.

If a process on another line tries to access data in a locked group, that process is paused until the file-save finishes saving all the items in that group and unlocks it.

If the (T) option is specified, the SAVE processor will prompt the user's terminal:

```
FILE-SAVE TAPE LABEL -
```

The response is written on the tape as part of the tape label.

10.29.1 MULTIPLE REEL SAVES

When the data to be saved exceeds the capacity of the mounted reel, a MOUNT NEXT REEL message appears on the terminal screen. A pound sign (#) prompt follows the message.

Remove the tape reel, which should have rewound itself. Mount and position the next reel to the BOT (Beginning Of Tape) mark. Make sure that the media is write-enabled and that the tape drive is on-line. Now enter one of the following characters at the #, as appropriate:

```
C - CONTINUE
O - OVERWRITE  (used in cases of erroneous tape labels)
Q - QUIT
```

This procedure also holds true for RESTORING multiple reels.

10.30 ACCOUNT-SAVE AND ACCOUNT-RESTORE

The system has the ability to save and restore single accounts. The ACCOUNT-SAVE PROC allows you to generate a save tape with only one account on it. The ACCOUNT-RESTORE verb is used to add a single account to an already existing PICK system.

ACCOUNT-SAVE PROC

The 'ACCOUNT-SAVE' PROC functions similarly to the 'FILE-SAVE' PROC. The files section contains no System Dictionary pointer or items, and only one account is saved. No synonym D or Q pointers will be saved. If STAT-FILE items are generated, they will pertain only to the saved account.

Account saves are performed as follows:

1. Log onto SYSPROG.

2. Mount a tape with a write ring.

3. Enter

   ACCOUNT-SAVE [CR]

4. The following is displayed:

   TAPE LABEL IF DESIRED

   Enter the text to appear as part of the tape label.

5. The following is displayed:

   ACCOUNT NAME?

   Enter the name of an account in the system dictionary.

ACCOUNT-RESTORE

An Account-restore can be performed from a save of a whole system or from an Account-save tape. In either case, the account will be restored and a pointer to the account will be created in the SYSTEM dictionary.

NOTE: The account must not already exist on the system.

Account-restores may be started from any tape of a multi-tape file-save! To save time in searching a tape, the STAT-FILE listing may be consulted to determine which reel the account's data starts on, and that reel may be mounted.

Account restores are performed as follows:

1. Log on to SYSPROG

2. Mount the tape with the account on it.

3. Enter

   ACCOUNT-RESTORE new-account-name [CR]

4. The following is displayed:

   ACCOUNT NAME ON TAPE?

   Enter the name of the account under which the account was saved.

5. The tape is searched for the account, and the restore proceeds automatically.

A 'Synonym' segment may be encountered with a base which has not been found on the tape. This would happen if a D pointer on the saved account pointed to a file on another account, or if a 'D' segment on the tape was unrecognizable because of a parity error. In this case, the message 'SYNONYM NOT FOUND'is displayed. The synonym D-pointer will not be created but the restore will continue.

RESTORE-ACCOUNTS

The RESTORE-ACCOUNTS PROC automatically restores all accounts from a FILE-SAVE tape that are not currently on the system. The account names are read from the tape itself and do not have to be entered. This PROC can be used, for example, after a system upgrade to restore all the user accounts.

To use the PROC, enter the following:

```
RESTORE-ACCOUNTS
```


10.31 SYSTEM STATUS: THE WHAT AND WHERE VERBS

The WHAT verb is used to display the system configuration, the current status of its locks and tables, and the location of the processes that are logged on. WHERE is used to display data for all lines that are logged on; the WHERE verb is a subset of the WHAT verb.

FORMAT:

WHAT {(options)}

The available options are

m{-n}	lines to display
'acct.name'	name of account whose status is to be displayed; the single quotes are required
L	suppresses display of all locks
P	directs output to the printer
S	suppresses the display of the spooler and printer information
W	suppresses display of the line status
Z	displays status of all lines including those not logged on

FORMAT:

WHERE {n} {(options)}

The available options are

m{-n}	lines to display
'acct.name'	name of account whose status is to be displayed; the single quotes are required
P	directs output to the printer
Z	displays status of all lines including those not logged on

The default form of the WHERE displays all lines that are logged-on. Some examples follow:

WHERE 3-5	Displays the return stack for users three through five.
WHERE 'DP'	Displays the return stack for all lines logged onto DP.
WHAT L	Suppresses the locks section.
WHAT LWS	Yields only system configuration section.

Forms of the WHAT and WHERE verbs.

The WHAT verb displays the state of the system as below (numbers in brackets are not part of display):

```
15:03:22  19 JAN 1988
CORE LINES PCB0 WSSTART WSSIZE SYSBASE/MOD/SEP  MAXFID  AVAIL. OVERFLOW
640K   17   512   1056    100     6156  11   1 97799            51234
       [1]  [2]    [3]    [4]      ...[5].....  [6]               [7]

15679 (3D3F)-11                                                    [8]

00 00 00 00 00 11 00 00 00 00 00 00 00 00 00 00                    [9]
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00                    [9]
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00                    [9]
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00                    [9]

00 00 00 00 00 00 00 00 00 00                                      [10]

 00 0200   7B30         6.15B 183.123 181.07F
 01 0220   7DB0        21.033   6.070   6.033  16.1C8  13.042
*02 0240   7F30       121.000 121.1AB 166.602
 07 02E0   7D30    T    6.070   6.033   5.054
 08 0300   7D30         6.070   6.033   5.054
 11 0360   5F30        11.156  16.0F4
 16 0400   3F30       165.083 164.04C
[11][12] [13][14] [15]...[16].................................

SPOOLER IS INACTIVE

PRINTER # 0 IS PARALLEL, INACTIVE, AND ON LINE.
THE PRINTER IS DEFINED AS PARALLEL PRINTER # 0.
ASSIGNED OUTPUT QUEUES:  0
THE NUMBER OF INTER-JOB PAGES TO EJECT IS 0.
The SPOOLER is in an unambiguous state.
```

NOTES:

[1] Number of communication lines (terminals) plus one (spooler) = number of processes on system.

[2] PCB-FID for line zero; each following line's PCB-FID is displaced by 32 frames from PCB0.

[3] Extended work-space starting FID; WSSTART = PCB0 + 32*LINES (including SPOOLER).

[4] Extended work-space size; there are 3 workspaces per line.

[5] System base-FID/modulo/separation; SYSBASE = WSSTART + WSSIZE*3*LINES.

[6] Maximum disk FID

[7] Available overflow space; linked frames + contiguous frames.

[8] Group-locks (if any); format= ddddd (xxxxx)-11 where: ddddd=group FID (decimal); xxxxx=group FID (hex); 11=line number

[9] PICK/BASIC execution locks; there are 64 execution locks and they start at location 127.A. If a lock is set, the line number of the process that set the lock is displayed.

[10] System locks; there are 10 system locks and they start at location 127.0

[11] Line number, preceded by a "*" if your line.

[12] PCB-FID (hex) of line.

[13] PIB-status of line:
7B/FB = Terminal output
5F = Waiting for disk
Typically, spooler is BF
7F/FF = Active, or ready to go
7D = Terminal input
3F = Release Quantum/Sleeping

[14] PIB-status-2: 00 - 70 = Normal; 80 - F0 = In DEBUGGER.

[15] T = Tape attached; P = Printer attached.

[16] Current location, followed by subroutine return-stack addresses with the most recently added address listed first; the format of the addresses is fff.lll where
fff = decimal FID; lll = hexadecimal offset into frame

FID addresses of some processes:

5	TCL
6-9	Terminal I/O
13-16	EDITOR
21	DEBUGGER
22-32	ASSEMBLER
53-70	ACCESS Compiler
71-100	LIST
107,109,180-189	BASIC
190-199	BASIC Compiler
200-220	File-save
290-298	RUNOFF
165	SPOOLER

For example, in the preceding example, lines 0, 7, and 8 are in terminal I/O processes; line 1 is in the debugger.

10.32 VERIFYING SOFTWARE

The VERIFY-SYSTEM PROC checks to see if the system software is correct.

The VERIFY-SYSTEM PROC generates a checksum for every frame of software from 1 to 511. These check-sums are compared with those in the ERRMSG file, in an item named "CHECK-SUM". This item contains the correct checksum for all the system software frames. Each line in the item contains a checksum for one frame of code, optionally followed by one or more hexadecimal limits. If the limits are present, the checksum is generated only for bytes within the limits. If no limits are present, the checksum is generated for bytes 0--X'7FF'. This is done because some frames contain tables which change from time to time, such as the system overflow table. Table entries are not checksummed, only assembly code.

If all the program frames verify, message 341 is printed:

```
[341] SYSTEM VERIFIED.
```

If a frame generates a checksum that does not match the checksum for that frame in the "CHECK-SUM" item, the FID of the frame, the generated checksum and the stored checksum from the item are printed, and message 342 is printed at the end of the check run:

```
[342] SYSTEM DOES NOT VERIFY!
THERE ARE n PROGRAM FRAMES WITH MISMATCHES!
```

where n is the number of programs whose check-sums do not match.

The VERIFY-SYSTEM PROC should be run whenever it is suspected that the system software is in error.

If a mismatch is found, the system software can be restored by performing an ABS restore. For information on ABS restore, see the Installation and Upgrade Guide.

√ PP64
6/27/88

Chapter 11

PICK PC-XT

THE PICK SYSTEM

USER MANUAL

PROPRIETARY INFORMATION

11 PICK PC-XT

11.1 PICK SYSTEMS IBM PC-XT PACKAGE 11-3
11.1.1 THE PICK OPERATING SYSTEM REQUIREMENTS 11-3
11.2 INSTALLATION AND BOOT GUIDE 11-4
11.2.1 DISK ALLOCATION 11-4
11.2.2 BOOTING FROM HARD DISK 11-4
11.2.3 BOOT/RESTORE FROM FLOPPIES 11-4
11.2.4 THE 'A' OPTION 11-5
11.2.5 THE 'F' & 'Q' OPTIONS 11-5
11.2.6 THE 'K' OPTION 11-6
11.3 FDISK VERB : FIXED DISK PARTITIONING 11-7
11.4 OFF-LINE STORAGE : FLOPPY AND TAPE 11-8
11.5 COLOR AND MONO VERBS : MEMORY MAPPED MONITOR 11-10
11.6 PRINT@ FUNCTIONS 11-12
11.7 TERM TYPES 11-14
11.7.1 IBM PC-XT LINE 0: TERM TYPE 11-14
11.8 IBM PC-XT KEYBOARD DIFFERENCES 11-15
11.9 SET-KBRD PROGRAM 11-16
11.10 SET-FUNC PROGRAM 11-16
11.11 SET-BAUD VERB : SETTING BAUD-RATE (Ports > 0) 11-17
11.12 POWER-OFF: CONTROLLED SHUT-DOWN 11-18
11.12.1 REBOOT 11-18
11.13 FORMAT: FORMATTING FLOPPY DISKETTES 11-19
11.14 EUROPEAN DATE FORMAT 11-19
11.15 DOS TO PICK BRIDGE : COPYDOS 11-20
11.15.1 THE M OPTION - MULTIPLE PICK ITEMS 11-21
11.15.2 THE T OPTION - TRANSLATING CHARACTERS 11-21
11.15.3 THE F OPTION - FLAG CHARACTERS 11-22
11.16 PICK TO DOS BRIDGE : COPYPICK 11-24
11.17 ABS EXTENSION 11-27
11.18 CREATING OR MODIFYING TERM TYPES 11-28
11.18.1 CREATE TERMINAL DEFINITION 11-28
11.18.2 MODIFY TERMINAL DEFINITION 11-29
11.18.3 DELETE TERMINAL DEFINITION 11-30
11.18.4 ADD TERMINAL TO SELECTED DEFINITIONS 11-31
11.18.5 DELETE TERMINAL FROM SELECTED DEFINITIONS 11-31
11.18.6 EXIT WITHOUT UPDATING SYSTEM-CURSOR 11-31
11.18.7 UPDATE SYSTEM-CURSOR TO SELECTED TERMINALS 11-31
11.18.8 DEFINING TERMININAL TABLES 11-31
11.18.9 TERMINAL TYPE 11-31
11.18.10 TERMINAL SIZE 11-32
11.18.11 CURSOR ADDRESSING TYPE 11-32
11.18.12 CURSOR CODE STRINGS 11-34
11.18.13 SPECIAL CURSOR CODE STRINGS 11-34
11.18.14 COLUMN ONLY CURSOR POSITIONING 11-35
11.18.15 CLEAR SCREEN & HOME @(-1) 11-35

11.1 PICK SYSTEMS IBM PC-XT PACKAGE

Enclosed in the PC-XT package are:

* Four (5) diskettes, labeled:
  - PICK PC SYSTEM #1
  - PICK PC SYSTEM #2
  - PICK PC SYSTEM #3
  - PICK PC DATA FILES #1
  - PICK PC DATA FILES #2
* PICK User Reference Manual
* End-user License Agreement
  (found on outside of package)

11.1.1 THE PICK OPERATING SYSTEM REQUIREMENTS

* IBM PC-XT. Either one or two hard disks are supported

* A minimum of 512K RAM memory.

* Either Monochrome or Color/Graphics monitor.

* Allocation of space for the PICK PC-XT System on the IBM hard disk must be one block of contiguous cylinders. Refer to Installation card for minimum cylinders required to install PICK

  Hard disk space is allocated by cylinders. On a standard IBM 10mb drive each cylinder has four (4) tracks. If each track has 8704 bytes, then an IBM cylinder has 34816 bytes.

  Optionally, one to 2 additional terminals and up to 3 IBM parallel printers may be connected.

11.2 INSTALLATION AND BOOT GUIDE

11.2.1 DISK ALLOCATION

When the PICK Operating System is installed it must locate a group of free and contiguous cylinders to claim for its partition space. If the other operating systems on the disk have already allocated the entire disk then they will have to relinquish space before a PICK partition can be created. In this case the other operating system(s) will have to be backed up, deleted and then reinstalled into smaller partitions so that space for PICK can be freed. Each operating system has its own utilities for installing itself, backing itself up and deleting itself.

Since PICK will install itself on all the remaining available hard disk space, if other operating systems are to co-reside with PICK, PICK must be installed last.

A minimum amount of free and contiguous disk space is required before PICK will install itself. The required space is measured in cylinders. The number of cylinders required is a function of the number of heads the hard disk drive has. Please refer to the PC-XT Installation Guide card for more details.

11.2.2 BOOTING FROM HARD DISK

If the PICK Operating System was marked as active (SEE: FDISK VERB) at the time the system was last shut off, PICK will boot from the hard disk when the system is powered up. When the boot process finishes, control is transferred to the COLDSTART procedure.

11.2.3 BOOT/RESTORE FROM FLOPPIES

Restoring the system means replacing the software and/or data on the hard disk from a backup media such as floppy diskettes.

Typically, software/data is restored when the running copy is suspected to to be damaged or when a new copy of the system is available.

To restore the Pick environment on the PC-XT, place PICK PC SYSTEM #1 diskette into the floppy drive and boot the system (CTRL-ALT-DEL).

Following the system sign-on message the screen will display:

OPTIONS: K)ill, A)BS only, F)ile & ABS, Q)uick file & ABS -

The 'A', 'F' and 'Q' options are used for restoring PICK.
The 'K' option is used to delete the PICK partition.

11.2.4 THE 'A' OPTION

The 'A' option will restore the PICK Operating System (sometimes refered to as the Monitor and ABS). Once you have entered an 'A' the following will occur:

1. The system will read #1, #2, and #3's contents and then go directly to the COLDSTART PROC just as if you had done a system boot. At this point the PICK Operating System has been restored. The accounts and data files, however, will not have been altered.

11.2.5 THE 'F' & 'Q' OPTIONS

The options 'F' and 'Q' initiate identical restore procedures. However, the 'F' option will reformat the PICK hard disk partition before starting the restore.

The 'Q' option skips the hard disk reformatting, hence 'Quick file & ABS'. When you have entered an 'F' or a 'Q' the following will occur:

1. PICK hard disk partition reinitialization. (F option only !)

2. The system will read #1, #2, and #3's contents and then prompt you to load a DATA FILE floppy:

   load PICK PC DATA FILES #1 then type 'C' to continue

   At this point you will do one of the following things:

   a. You will load your #1 FILE-SAVE diskette, if you want to restore the accounts and data files as they exist on your last FILE-SAVE.

   b. You will load the PICK PC DATA diskette if you want to restore the accounts and data files as they were when you first installed the Pick system.

If loading FILE-SAVE disks, you will be prompted to load diskettes as necessary by the following:

load PICK PC DATA FILES #2 then type 'C' to continue
LABEL dd mmm yyyy acct.name PICK PC-XT release.revision #
At this point mount the next data diskette in the drive and type 'C' to continue.

5. When the system has read the data diskette(s) it will go directly to the COLDSTART PROC just as if you had done a system boot. At this point the PICK Operating System will have been restored, a full file-restore accomplished and you will be in the PICK environment.

11.2.6 THE 'K' OPTION

The 'K' option will delete, or kill, the PICK partition on the hard disk(s). When you have entered a 'K', the following will occur:

The screen will display the message: "Are you sure (Y or N)". If the operating system is to be deleted, key in 'Y'. WARNING --- this will delete all data from the PICK partition. If you don't want the operating system deleted, key in 'N'. The system will re-display the OPTIONS (A, F, K , Q) prompt.

11.3 FDISK VERB : FIXED DISK PARTITIONING

The verb FDISK has been provided to support the use of IBM's Fixed Disk Partitioning concept.

FORMAT:

>FDISK (from SYSPROG account)

IBM's Fixed Disk Partitioning concept allows up to four (4) different operating systems to co-reside on one or two drives. They each "live" on the hard disk within their own range of contiguous cylinders. The FDISK verb or command, as implemented under PC DOS and PICK, allows the user control over which operating system is currently executing. FDISK allows a user to mark one of the co-resident operating systems as "active". When the system is booted, the operating system marked as "active" assumes control of the machine. Therefore, to move from operating system "A" to operating system "B", FDISK is invoked, "B" is marked "active" and when the system is booted system "B" will be active.

The PICK implementation of its FDISK verb closely resembles IBM's PC DOS FDISK command in both presentation and capabilities.

For users familiar with PC DOS' FDISK, PICK's FDISK differs in two ways:

1. The create a partition option is non-functional under PICK because partition creation is handled automatically at system installation time.

2. The delete partition option will refuse to delete the PICK partition unless another partition is first made "active". In the case where PICK is the only operating system this does not apply.

3. Deleting the PICK partition from drive C will also delete PICK from drive D, if it exists. With FDISK, the user may view but not modify the partitions on drive D.

11.4 OFF-LINE STORAGE : FLOPPY AND TAPE

The SET-SCT and SET-FLOPPY verbs are used to set up the peripheral storage devices.

The PC-XT can use either the floppy diskette drive or a supported 1/4" streaming cartridge tape (SCT) as a peripheral storage device. The desired device is assigned to your line with a SET-FLOPPY or a SET-SCT command.

Streaming Cartridge Tape (SCT)

FORMAT:

SET-SCT (BLK-SIZE)

This command sets the peripheral storage device to SCT 1/4" tape. The SET-SCT command also does an automatic tape attach (T-ATT) of the SCT to your line. The BLK-SIZE is set to 16384 as the default size or it can be re-specified to values from 2048 to 16384.

Additional SCT commands:

T-RETEN

This command re-tensions the SCT by first forward and then backspacing the tape its entire length (no data is destroyed). It is recommended that tapes be re-tensioned before they are used to reduce the occurance of parity errors.

T-ERASE

This command also re-tensions the SCT by first forward and then backspacing the tape its entire length. In this case, however, the entire tape is erased.

T-STATUS

This command returns with drive and type information of the currently assigned peripheral storage device.

T-REW

This command rewinds the tape to the beginning from its current position. It is important to do a T-REW after write operations such as T-DUMPS before the tape is removed from the drive. This is because the T-REW writes a terminating EOF mark before rewinding the tape.

Floppy Diskette

FORMAT:

SET-FLOPPY ((density,drive)

This command sets the tape device to a floppy drive.
If no options are specified, drive A is the default.

Options for the density parameter are as follows:

S - Standard Density - 360 KB

L - Low Density - 320 KB

Options for the drive parameter are as follows:

A - Floppy Drive A

B - Floppy Drive B

SET-SCT	Set tape device to 1/4" tape.
SET-FLOPPY (SB	Set tape device to floppy drive B, standard density
SET-FLOPPY (A	Set tape device to floppy drive A.
SET-FLOPPY	Set tape device to floppy drive A.

Sample usage of SET-SCT and SET-FLOPPY commands.

Note: Tapes written with Pick data must first be erased prior to being used with DOS.

See also the PC PERIPHERAL INSTALLATION GUIDE, found in the back of this manual, for more information.

11.5 COLOR AND MONO VERBS : MEMORY MAPPED MONITOR

Two verbs, COLOR and MONO, have been provided to support the use of IBM's memory mapped monitors. In addition, PICK/BASIC and PROC now support IBM's memory mapped monitors.

FORMAT:

COLOR (foreground color)(,background color)(switches)

The supported colors for background and foreground are:
Black, Blue, Green, Cyan, Red, Magenta, Brown and White

The supported switches are:

/B or /BLINK	Activate character blinking
/NB or /NOBLINK	De-activate character blinking
/F or /FULL	Full intensity foreground
/H or /HALF	Half intensity foregound
/R or /REVERSE	Activate reverse video
/NR or /NOREVERSE	De-activate reverse video

STATEMENT	EXPLANATION
COLOR RED	Set foreground to red.
COLOR ,BLUE	Set background color to blue.
COLOR /B	Activate character blinking.
COLOR /NB/R	Deactivate character blinking, activate reverse video.
COLOR BROWN/F	Set foreground to brown and full intensity.
COLOR ,RED/H	Set background to red, set foreground to half-intensity.
COLOR GREEN,CYAN/HALF	Set foreground to half intensity green, background cyan.

Sample usage of the COLOR verb.

The COLOR verb only works on line 0. It will refuse to execute if a COLOR/GRAPHICS adapter is not in the system.

FORMAT:

MONO (switches)

The supported switches are identical to the COLOR verb switches, with the addition of the following:

/U or /UNDERLINE	Activate character underlining
/NU or /NOUNDERLINE	Deactivate character underlining

The MONO verb only works on line 0. It will refuse to work if a monochrome adapter card is not in the system.

11.6 PRINT@ FUNCTIONS

New functions have been added to the PICK/BASIC "PRINT_@" statement and the PROC "T" command to support IBM's memory mapped monitors.

The "PRINT @" statement and the PROC "T" command formerly allowed negative integers in the range -1 to -10 as arguments. For the PICK PC-XT implementation the argument range has been extended from -1 to -127.

For all PICK machines the ranges break down as follows:

-1 TO -32	Functions which affect all machines
-33 to -127	Functions which are implementation specific

More specifically:

(across all future PICK implementations)

-1 to -10	Remain as defined in the PICK Refernce Manual
-11 to -16	Are defined in the DOC file on SYSPROG
-17 to -32	Are reserved for future expansion

(for the IBM PC-XT implementation)

-33 to -40	Define background colors
-41 to -48	Define full intensity foreground colors
-49 to -56	Are reserved for future expansion
-57 to -64	Define half intensity foreground colors
-65 to -88	Are reserved for future expansion
-89 to -96	Define IBM memory mapped monitor modes
-97 to -127	Are reserved for future expansion

PICK/BASIC Examples:

```
* Clear screen and home the cursor
PRINT @(-1)

* Clear screen and home the cursor (same, but more readable)
CLEAR.SCREEN = _@(-1)
PRINT CLEAR.SCREEN
```

PROC Examples:

```
PQ
001 C Clear the screen and home the cursor
002 T (-1)

PQ
001 C Activate Color/Graphics, foreground = blue, background = white
002 T (-93),(-63),(-49)
```

NOTE: In the SYSPROG account in the file SYSPROG-PL is a program called DEMO which demonstrates the extended support for memory mapped monitors under PICK/BASIC.

11.7 TERM TYPES

11.7.1 IBM PC-XT LINE 0: TERM TYPE

Term type "I" is for the IBM memory-mapped monitors; monochrome or color/graphics. This term type is only to be used on line 0.

The following is a table of other active term types for the PC-XT:

A	ADDS 580
B	AMPEX 210
C	CITOH VT52
D	DATAMEDIA
E	ESPRIT
G	IBM 3161/3163
I	IBM PC-XT (port 0)
L	LEAR SIEGLER
M	AMPEX 80
P	PERTEC 701
R	ADDS REGENT
T	TELEVIDEO 910
V	ADDS VIEWPOINT
W	WYSE 50

PC-XT Term Types

"TERM n" can be typed on any port to invoke the term type that matches the terminal attached to that line.

A BASIC program called 'TERM-TYPE' can be used to pre-set all term types on your system. See the 'ADDENDA' file in SYSPROG for more information on the TERM-TYPE program.

Terminal types can be created and/or modified using the 'DEFINE-TERMINAL' command. See section 11.19 for instructions.

11.8 IBM PC-XT KEYBOARD DIFFERENCES

Certain capabilities of the IBM System keyboard are disabled under the PICK Operating System to make the keyboard appear to be a standard CRT keyboard.

The following is a list of keyboard changes under PICK:

- Keypad area now generates numerics only.
- Backtabs cannot be generated from the keyboard.
- Print screen functions disabled.
- The Function Keys F1 to F10 are user-definable via the included BASIC program SET-FUNC.
- The ALT key recognition is disabled except in these cases: the CTL-ALT-DEL keyboard reset sequence, the ALT nnn special ASCII character generation sequence, and the ALT key in combination with user-defined function key.

11.9 SET-KBRD PROGRAM

New SET-KBRD program allows line 0's keyboard to be redefined.

FORMAT:

>SET-KBRD <filename> <itemname>

Where <filename> is the name of the file which holds the keyboard definition item <itemname>.

Several keyboard definition items are included with the system. These may be found in the file KEYBOARDS in the account SYSPROG. These include:

FRENCH/FRENCH 2
GERMAN
SPANISH/SPANISH 2
ITALIAN
ENGLISH
USA
DVORAK

The system 'boots-up' using the USA keyboard.

You may want to create your own keyboard definition items. An explanation of how to create such definitions is found in the SYSPROG ADDENDA file.

11.10 SET-FUNC PROGRAM

New SET-FUNC program allows the line 0 function keys to be redefined.

FORMAT:

>SET-FUNC <filename> <itemname>

Where <filename> is the name of the file which holds the function key definition item <itemname>.

Some example function key definition items are included with the system. These may be found in the file FUNCKEYS on the SYSPROG account.

The system 'boots-up' with the function keys undefined. If you want to create your own set of function key definitions, there is an explanation of how to do so in the SYSPROG ADDENDA file.

11.11 SET-BAUD VERB : SETTING BAUD-RATE (Ports > 0)

The SET-BAUD verb allows serial ports to be set to various baud rates.

FORMAT:

>SET-BAUD (line#),baud-rate

The SET-BAUD verb only effects serial ports. If the line number parameter is not present, the line that you are logged onto will be used.

Meaningful baud-rates are listed below.

50	forced to 110
75	forced to 110
110	
134.5	forced to 150
150	
300	
600	
1200	
1800	forced to 2400
2000	forced to 2400
2400	
3600	forced to 4800
4800	
7200	forced to 9600
9600	

Meaningful Baud-Rates.

NOTE: With only 1 serial I/O port, it must be configured as line 1 for SET-BAUD to work. If a sole serial I/O port is set up as line 2, then the SET-BAUD will not see it.

STATEMENT	EXPLANATION
SET-BAUD 1,4800	Sets port 1 to 4800 baud.
SET-BAUD 2,1800	Sets port 2 to 2400 baud.

Sample usage of the SET-BAUD verb.

11.12 POWER-OFF: CONTROLLED SHUT-DOWN

The use of this verb prior to system power-down ensures that all write-required frames are flushed from memory to disk.

FORMAT:

>POWER-OFF

The operating system automatically flushes memory buffers to disk whenever the system has been quiescent for two (2) seconds. If this automatic flush has already taken place, the system can be powered down without having to type POWER-OFF.

POWER-OFF will disable all users, flush memory to disk, and put the machine into a HALTed state from which powering off and then back on is the only recovery. POWER-OFF only functions if all other users are logged off. If they are not, you will be informed who is still logged on and returned to TCL.

Using the POWER-OFF verb is a recommended procedure.

This verb only works on Line 0.

11.12.1 REBOOT

REBOOT will allow the PICK System to re-load the operating system without doing normal BOOTing procedures.

FORMAT:

>REBOOT

A new verb, REBOOT, exists in the Master Dictionary of the SYSPROG account. REBOOT will disable all users, flush memory to disk, and cause the system to boot, just as if (ALT-CTRL-DEL) had been pressed. REBOOT only functions if all other users are logged off. If they are not, you will be informed who is still logged on and returned to TCL.

This verb only works on line zero (0).

11.13 FORMAT: FORMATTING FLOPPY DISKETTES

This program formats floppy diskettes under the PICK Operating System.

FORMAT:

>FORMAT **(from SYSPROG account)**

This program formats diskettes in 9 sectored, double-sided format. This is consistant with IBM's current standard. These diskettes, however, are NOT usable under PC or MS DOS, because PICK's formatter program does not build the necessary DOS File Allocation Tables (FAT).

To be useable by PICK, diskettes must format "perfectly". No bad sectors are allowed. The PICK formatter will display an error message when a bad diskette is encountered.

11.14 EUROPEAN DATE FORMAT

The European date format is supported on the PC-XT.

FORMAT:

>SET-DATE-EUR

Sets the system to use European date format

>SET-DATE-STD

Sets the system back to non-European data format.

This can be tested by keying in and using the following PICK/BASIC program:

```
001 PRINT DATE() 'D/'
002 END
```

EXAMPLE:

```
>SET-DATE EUR
>SET-DATE              Enter date as:  DD/MM/YY

>SET-DATE-STD
>SET-DATE              Enter date as:  MM/DD/YY
```

11.15 DOS TO PICK BRIDGE : COPYDOS

The COPYDOS verb allows data contained in the DOS partition to be transferred into the PICK partition.

FORMAT:

```
>COPYDOS dospath ((options()))
TO:(filename (itemname)
```

The syntax for the dospath parameter is as follows:

```
drive:\{subdirectory\...}dosfilename
```

For example:

```
drive:\subdirectory1\subdirectory2\dosfilename
                or
drive:\dosfilename
```

OPTIONS	MEANING
O	Overwrite existing PICK item.
S	DOS data is in sequential file mode. (default)
R	DOS data is in random file mode.
T	Translate characters. System prompts for specifics.
F	Flag characters. System prompts for specifics.
M	Make Multiple items. System prompts for specifics.

If neither S nor R is specified in the options, the S(equential) option is assumed.

If neither the T nor F option is specified, the COPYDOS process will translate the DOS character X'0D' to X'FE'. This causes every DOS line delimited with a carriage return, to become a PICK attribute. The linefeed X'0A' and null X'00' characters are deleted.

The T, F and M options are explained further in the following section.

If the optional PICK itemname is not input, following the ' TO:(' prompt, the DOS filename (from the dospath parameter) is used as the PICK itemname.

Note also, that if the DOS file is larger than 25000 bytes and the M option has not been specified, the COPYDOS process will automatically split the file into PICK items and assign the names itemname0, itemname1, etc.

11.15.1 THE M OPTION - MULTIPLE PICK ITEMS

The M option allows for regulation of the size of the targeted PICK items. This flexibilty can be very useful in aligning DOS data with PICK attributes.

After entering a response to the ' TO:(' prompt the system displays:

ENTER LENGTH OF RECORD (OR Dn):

The user may elect to enter a numeric response indicating how many bytes each data portion of the PICK item will be, or a 'D' followed by a number to indicate how many PICK attributes (lines) each PICK item will be.

Using the 'Dn' response assumes that the default translate of carriage return/line feed to attribute marks will be performed.

When the DOS file is converted to multiple items, the items will be created with boundries established by attribute marks, provided that they exist in the translated file. Should no attribute marks exist, items will contain a single attribute with the specified number of bytes.

The PICK item-ids are generated by concatenating the itemname used in the ' TO:(' specification with 0 , 1 , 2 , etc. No new PICK item with just the itemname alone will be created.

Entering a <CR> after the appropriate response causes the COPYDOS process to begin or, if an F or T option is in effect, further prompting as noted below.

11.15.2 THE T OPTION - TRANSLATING CHARACTERS

The 'T' option is available to translate up to 16 different hex bytes. Upon entering a response to the ' TO:(' prompt, the system will display:

REPLACE:

After entering the hex character to replace, the system displays:

WITH:

Entering identical hex strings in response to both REPLACE: and WITH:, causes that character to be deleted from the file.

After entering a response to the WITH: prompt, the system will again display:

REPLACE:

Entering a <CR> after the REPLACE: prompt, terminates further hex character prompting, and the system displays:

OKAY(Y/N):

An 'N' response allows the user to re-enter all of the Translate specifications. A 'Y' response causes the COPYDOS process to begin.

11.15.3 THE F OPTION - FLAG CHARACTERS

The 'F' option is available to Flag up to 16 different hex bytes by placing a hex'00' in front of the Flagged hex character.

Entering in different hex bytes in response to the 'REPLACE:' and 'WITH FLAGGED' prompts, results in a Translation, with the resultant PICK byte preceded with hex'00'.

Entering in the same hex byte in response to both prompts, results in passing that byte unchanged and preceding it with hex'00'. Note that this is in contrast with the T option, where specifying the same character, deletes that character.

If an 'F' option is in effect, upon entering the ' TO:(' response the system will display:

REPLACE:

After entering the hex character to translate or pass, the system displays:

WITH FLAGGED:

Upon entry of the same character or a replacing character, the system will prompt for up to 15 additional hex characters to Flag. The system will prompt for additional characters, until a <CR> is entered at the REPLACE: prompt. A <CR> causes the system to display:

OKAY(Y/N):

An 'N' response allows re-entry of the FLAG specifications.

In the following example session, note that whenever multiple items are produced, (even without the M option), that an item with the actual PICK itemname used in the specification does not exist. The first item is the specified itemname with a zero appended.

```
>COPYDOS C:\SUB1\SUB2\DOSFILE (SMT

 TO:( PROCLIB PICK.SIDE


ENTER LENGTH OF RECORD (OR Dn):  D5

   REPLACE:  0D                  WITH:  FE
   REPLACE:  0A                  WITH:  0A
   REPLACE:  2C                  WITH:  2A
   REPLACE:  <CR>

   OKAY(Y/N):  Y

   READING DIRECTORY
          SUB1
          SUB2
          DOSFILE
   WRITING ITEM
          PICK.SIDE0
          PICK.SIDE1
          PICK.SIDE2
          PICK.SIDE3
  END OF FILE
```

Sample usage of the COPYDOS utility with the S, M and T option.

11.16 PICK TO DOS BRIDGE : COPYPICK

The COPYPICK utility allows data contained in the PICK partition to be transferr into the DOS partition.

Since a PICK to DOS transfer is done from a DOS partition, a DOS disket (available from your PC dealer) contains this utility. It is loaded with t command:

```
COPY A:*.* C:
```

This loads the PICK to DOS bridge program, COPYPICK.EXE into the DOS drive C:

FORMAT:

```
C>COPYPICK
```

Upon entering a <cr> the system will first prompt for:

```
Options:
```

Option	Meaning
I	Include Item-id as a line within the DOS file. The Item-id is alwa preceeded with a line feed character. This option may be of particul use when moving all items in a file with the intent of doing addition processing in the DOS environment. The beginning of an item may detected by the presence of a line feed either at the beginning of t file or immediately following another line feed.
N	Numeric Item-id's are assumed. This will allow all items within a file be moved to a DOS file in numeric order. Rather than a prompt for t Item-id, a range is requested. All items within this range are access out of the specified file. Should an item not be present, it is ignor and the process continues.
D	Diagnostic mode operation has two levels. If the letter D is the fir character of the option position string, a summary set of diagnost messages is displayed. If the letter D is in option position 2 greater, a more detailed set of diagnostics is presented.

After the Options request, the system will then prompt for:

```
PICK Account Name:

PICK File (DICT FILE or FILE or FILE,DATA):

DOS File
```

PICK Item or * for all:

A successful transfer ends with the following message:

n item(s), n records/attributes transferred

The DOS file name may be any valid DOS file on any valid DOS device. The DOS prompt occurs before the request for the Item-id's so several items may be store the same DOS file. This is done by specifying individual Item-id's. A null It denotes the end of input and precludes transfering an item with a null item-id. asterisk '*' is reserved for transfering all items within a file. This means t single item with an Item-id of asterisk cannot be transfered.

When transfering single items, the message displaying the number of items trans and the number of records/attributes will accumulate and display the totals trans to the DOS file.

If an invalid name is entered at any time during the prompt sequence, an approp error message, such as the following, is displayed:

File xxx in account yyy not found

After a successful transfer, the user will be prompted for another PICK filena <cr> response causes a prompt for a different PICK account name. A <cr> at the ac prompt exits the COPYPICK utility.

PICK attribute marks (X'FE') are replaced with a carriage return (X'0D') and line (X'0A'). This effectively makes each PICK attribute value, a DOS record in the receive file.

```
C> COPYPICK

   Options:
   PICK Account Name: SYSPROG
   PICK File (DICT FILE or FILE or FILE,DATA): BP
   DOS File FINDX
   PICK Item or * for all: FIND

   1 Item(s) Converted
   47 Attributes Moved

   PICK Item or * for all:

   1 Item(s) Converted
   47 Attributes Moved

   PICK File (DICT FILE or FILE or FILE,DATA):
   PICK Account Name: <cr>

C>
```

Sample COPYPICK transfer session.

11.17 ABS EXTENSION

The size of the ABS area may be increased to accomadate additional assembly level requirements.

NOTE: IGNORE THIS SECTION UNLESS YOU FALL INTO ONE OF THE FOLLOWING CATEGORIES:

1. Your application is comprised of Assembler programs that exceed a total of 100 frames.

2. Your Assembler program development is anticipated to require more than 100 frames.

During the intial virgin boot procedure, a proprietary message is displayed. There is a three (3) second window following this proprietary message. Three percent signs are displayed, about one per second. While these per-cent signs are appearing, the character 'A' is entered from the keyboard. The following is displayed:

```
%%%                    <------- enter 'A' before 3rd %

 Enter total ABS frames (4095 >= #ABS >= 512) =
```

The number of ABS frames may be increased in increments of 32 frames.

The ABS area must reside totally on the first hard disk. (4096 ABS frames uses 8MB of disk.) The # of cylinders in the disk allocation table must be adjusted accordingly to reflect any additional ABS space used.

This extension should not be done unless absolutely necessary. Needless ABS extension negates available disc space.

NOTE: When calling for technical support, your first words should be that you have an 'ABS EXTENDED SYSTEM'.

11.18 CREATING OR MODIFYING TERM TYPES

"The DEFINE-TERMINAL" utility can be used to create or modify term types.

FORMAT:

>DEFINE-TERMINAL (from SYSPROG account)

The DEFINE-TERMINAL utility provides the user a means to customize the terminal characteristcs functions required for his particular needs. The utility provides an editor, selection process and "compiler". The utility is a menu-driven BASIC program which creates, maintains, compiles and selects up to 26 different terminals for inclusion in the table. Although only 26 terminals may be selected for inclusion in this utility, any number of terminals may be defined by this utility.

Upon entering the command "DEFINE-TERMINAL" at TCL, a display and menu similar to the following will be presented:

```
System Cursor Definition Utility

 The following terminals are defined.  Terminals marked with an asterisk
(*) are selected to be included in your System Cursor Definition.

*A ADDS            H HONEYWELL     *N WYSE100      *V VIEWPOINT
 B BEEHIVE         I IBM3010       *Q MIME         *W WYSE50
 C DTC            *J VT100         *R REGENT        X DATAGRAPHIX
 D DATAMEDIA      *K VT52          *S SOROC
 E EMULOG200      *L LSI            T TEC
 G GTC            *M AMPEX         *T TV920

     1)  Create Terminal Definition
     2)  Modify Terminal Definition
     3)  Delete Terminal Definition
     4)  Add Terminal to Selected Definitions
     5)  Delete Terminal from Selected Definitions
     EX  Exit without updating System-Cursor
     FI  Update System-Cursor to selected terminals

         Enter Selection (1-5) or EX or FI:
```

In the following sections, each of the menu choices will be explained.

11.18.1 CREATE TERMINAL DEFINITION

This choice (1) will allow the creation of a new terminal definition. A terminal definition consists of a series of parameters which the system requires to control a particular terminal (type, size, control codes, etc.).

After entering the menu selection (1), the routine will prompt for the terminal name to be defined.

It will then check if that name already exists. If so, you may opt to modify the existing definition, or enter another name. If you opt to modify the existing definition, then the routine will proceed as in the next section (modification). If the name is new, then you will be asked if you want to use a copy of an existing terminal definition for the intial values for the new definition. If so, you will be prompted for the name of the existing terminal to be used as a "template". This is useful for defining terminals which are similar to other existing terminals. If you do not choose to use an existing terminal definition as a "template", then the routine proceeds to prompt for each of the parameters for the new definition.

Otherwise, the routine proceeds as in the next section, modification.

11.18.2 MODIFY TERMINAL DEFINITION

This choice (2) will allow the modification of existing terminal definitions. After entering the menu selection (2), the routine will prompt for the name of the terminal definition to be modified. If the name does not exist, you may opt to create it. If you opt to create a new definition, then the routine proceeds as in the previous section (new definition). Otherwise, the routine proceeds to the definition modification mode.

In the terminal definition modification mode, the set of parameters for the terminal is broken into page size blocks for display and modification. First, a section of the existing definition is displayed, then the prompt "Modify Lines?" is issued. You may answer Yes, No (default) or a list of line numbers to modify (if you answer Yes, you will be prompted for the list of line numbers). If you answer No, then the next section of the existing definition is displayed and the process repeats until the entire definition has been reviewed. Otherwise, you will be prompted for each of the selected lines to enter new data. At each of these prompts, the folowing special entries may be made: First, a carriage return (null value) will cause the data for the line to be unchanged. Second, entering any number of spaces will cause the data for the line to be changed to null. Third, entering a single questions mark (?) will present a brief explanation of the contents of the line, and then reprompt for input. After all the lines have been prompted for, then the routine returns to the display section to again review the selection.

Once all the sections have been reviewed, you will be asked if the terminal definition is correct. If not, then the review and modify process will be repeated, or you may exit without saving any modifications.

If the definition is correct, then an attempt will be made to "compile" it.

If the compilation detects errors, then you may have to correct the errors via the modification process. Otherwise, you may select the terminal to be included in the list of terminals for your System-Cursor.

The following is an example of the display portion of the modification mode:

```
Terminal = TV920

   1.  TYPE.................................. T
   2.  SCREEN SIZE........................... 80,24
   3.  CURSOR ADDRESS CODE................... L
   4.  @(X) CURSOR POSITIONING............... CR STR(CHAR(12),X)
   5.  @(X,Y) CURSOR ADDRESSING.............. ESC "=" Y X
   6.  @(-1) CLEAR SCREEN & HOME............. CHAR(26)
   7.  @(-2) CURSOR HOME..................... CHAR(30)
   8.  @(-3) CLEAR TO END OF PAGE............ ESC "Y"
   9.  @(-4) CLEAR TO END OF LINE............ ESC "T"
  10.  @(-5) START BLINK..................... ESC " "
  11.  @(-6) STOP BLINK...................... ESC "q"
  12.  @(-7) START PROTECT................... ESC ")"
  13.  @(-8) STOP PROTECT.................... ESC "("
  14.  @(-9) CURSOR BACK.................... BS
  15.  @(-10) CURSOR UP..................... VT
Modify lines? NO
Terminal = TV920

  16.  @(-11) SLAVE ON.....................
  17.  @(-12) SLAVE OFF....................
  18.  @(-13) START REVERSE VIDEO........... ESC "j"
  19.  @(-14) STOP REVERSE VIDEO............ ESC "k"
  20.  @(-15) START UNDERLINE............... ESC "l"
  21.  @(-16) STOP UNDERLINE................ ESC "m"
  22.  @(-17) ENABLE PROTECT MODE........... ESC "&"
  23.  @(-18) DISABLE PROTECT MODE.......... ESC "'"
  24.  @(-99) EMBEDED VISUAL ATTRIBUTES?.... YES

Modify lines? NO

Is table for terminal TV920 correct? YES
```

11.18.3 DELETE TERMINAL DEFINITION

This choice (3) allows for the deletion of terminal definitions. After entering the menu choice (3), you will be prompted for the name of the terminal to be deleted.

11.18.4 ADD TERMINAL TO SELECTED DEFINITIONS

This choice (4) allows for the addition of a terminal to the list of terminals to be included in your System-Cursor. After entering the menu selection (4), you will be prompted for the name of the terminal to be added to the list of selected terminals. If the name exists, then the routine will check if that type of terminal (terminal type in the definition) has already been selected.

If not, then the desired terminal will be selected.

If the type of the desired terminal has already been selected for another terminal, then you will be asked if you want to replace the previous selection with the new selection. If so, the previous selection will be deleted from the list of selected terminals, and the new selection added.

11.18.5 DELETE TERMINAL FROM SELECTED DEFINITIONS

This choice (5) allows for the deletion of a terminal from the list of terminals to be included in your System-Cursor.

11.18.6 EXIT WITHOUT UPDATING SYSTEM-CURSOR

This choice (EX) quits the defintion process without updating the operating System-Cursor. All the modifications and selections made are preserved.

11.18.7 UPDATE SYSTEM-CURSOR TO SELECTED TERMINALS

This choice (FI) quits the definition process after updating the operating System-Cursor to the new selections.

11.18.8 DEFINING TERMININAL TABLES

The Terminal Definition Utility is used to define the Terminal Tables (menu choices 1 and 2). Some of the fields in the Terminal Table which are required are explained here.

11.18.9 TERMINAL TYPE

The terminal type is a single upper-case letter which identifies the terminal to the system. The terminal type field in the terminal table corresponds to the type as set with the TERM command.

11.18.10 TERMINAL SIZE

The size field contains the screen size in columns and rows. The size is entered as two numbers separated by a comma (e.g., 80,24). If a value exceeds the size, then the maximum size is substituted.

11.18.11 CURSOR ADDRESSING TYPE

The cursor addressing type is usually a single letter (A, L, T, H, D). The defined types are "A" for ADDS type addressing, "L" for Lear-Seigler type addressing, "T" for TEC type addressing, "H" for Hazeltine type addressing and "D" for decimal type addressing.

All types except "D" produce binary column and row addresses (single byte for each). "D" type addressing produces one to three digits for column and row addresses. If "D" type addressing is used, the code may be followed by two digits (22, 23, 32, 33) to force padding to the desired number of digits (e.g., "D32" will produce decimal addressing with 3 digits used for the column and 2 digits for the row (leading zeros added to force the length). "D" alone will use "floating" decimal numbers from 1 to 3 digits.

All cursor addressing codes may be followed by a plus sign "+" which adds one to the column and row addresses before generating the address codes. This allows for terminals which define the upper-left corner of the screen as "1,1" instead of "0,0". Thus, decimal addressing with a three digit row and column address numbered from "1,1" would be: "D33+".

To determine the proper binary cursor addressing type (A, L, T, H), use the table provided on the next page. This table shows the column or row address, and the associated code.

```
ADDS    COL = CHAR((INT(X/10)*6)+X)
        ROW = CHAR(Y+64)

TEC     COL = CHAR(-(1+X))
        ROW = CHAR(-1(+Y))

LSI     COL = CHAR(X+32)
        ROW = CHAR(Y+32)

HAZE    COL = CHAR(X)
        ROW = CHAR(Y)
```

X	Y	ADDS COL	ADDS ROW	LSI	TEC	HAZE	X	Y	ADDS COL	ADDS ROW	LSI	TEC	HAZE
0	0	nul	@	space	del	nul	40	@			H	W	(
1	1	soh	A	!		Soh	41	A			I	V	)
2	2	stx	B	"		stx	42	B			J	U	*
3	3	etx	C	#		etx	43	C			K	T	+
4	4	eot	D	$		eot	44	D			L	S	,
5	5	eng	E	%	z	eng	45	E			M	R	-
6	6	ack	F	&	y	ack	46	F			N	Q	.
7	7	Bel	G	'	x	bel	47	G			O	P	/
8	8	bs	H	(	w	bs	48	H			P	O	0
9	9	ht	I	)	v	ht	49	I			Q	N	1
10	10	dle	J	*	u	lf	50	P			R	M	2
11	11	dc1	K	+	t	vt	51	Q			S	L	3
12	12	dc2	L	,	s	ff	52	R			T	K	4
13	13	dc3	M	-	r	cr	53	S			U	J	5
14	14	dc4	N	.	Q	so	54	T			V	I	6
15	15	nak	O	/	p	si	55	U			W	H	7
16	16	syn	P	0	o	dle	56	V			X	G	8
17	17	etb	Q	1	n	dc1	57	W			Y	F	9
18	18	can	R	2	m	dc2	58	X			Z	E	:
19	19	em	S	3	1	dc3	59	Y			[	D	;
20	20	space	T	4	k	dc4	60					C	<
21	21	!	U	5	j	nak	61	a			]	B	=
22	22	"	V	6	i	syn	62	b				A	>
23	23	#	W	7	h	etb	63	c				@	?
24		$		8	G	can	64	d				?	@
25		%		9	F	em	65	e			a	>	A
26		&		:	e	sub	66	f			b	=	B
27		'		;	d	esc	67	g			c	<	C
28		(		<	c	fs	68	h			d	;	D
29		)		=	b	gs	69	i			e	:	E
30		0		>	a	rs	70	p			f	9	F
31		1		?		Us	71	q			g	8	G
32		2		@		space	72	r			h	7	H
33		3		A		!	73	S			i	6	I
34		4		B	]	"	74	t			j	5	J
35		5		C		#	75	u			k	4	K
36		6		D	[	$	76	v			l	3	L
37		7		E	Z	%	77	w			m	2	M
38		8		F	Y	&	78	x			n	1	N
39		9		G	X	'	79	y			o	0	O

11.18.12 CURSOR CODE STRINGS

The cursor code strings are expressions which produce the control and escape sequences used by the terminal being defined. The expressions are similar to BASIC syntax, except that a blank may be used between elements in the expression as well as a colon. Cursor code strings may consist of the following separated by blanks or colons:

1) Defined control character (e.g., ESC, BS, DEL, etc.)
2) String literal in quotes (e.g., "A", '[0', etc.)
3) Character function (e.g., CHAR(21))
4) Hexidecimal string (e.g., HEX(1B41))
5) String function (e.g., STR(NUL,5) or STR(CHAR(12),X00
6) Cursor address variable (e.g., X, Y, or Z)

The cursor address variables (X, Y, Z) cause the specified address (byte or decimal string) to be inserted into the control string at the specified position. The variable X contains the column, Y contains the row, and Z contains the row previously referenced in an @(X,Y) code (or zero if the last reference was @(-1) or @(-2).

The symbolic name for the control codes and their decimal and hexidecimal equivalents are shown in the table below. Any of these codes may be included in the cursor code string. It is often easier to reference the backspace character as BS instead of CHAR(8), or NUL instead of CHAR(0).

CODE	DEC	HEX	CODE	DEC	HEX	CODE	DEC	HEX
NUL	0	00	DLE	16	10	SP	32	21
SOH	1	01	DC1	17	11	DEL	127	22
STX	2	02	DC2	18	12			
ETX	3	03	DC3	19	13			
EOT	4	04	DC4	20	14			
ENQ	5	05	NAK	21	15			
ACK	6	06	SYN	22	16			
BEL	7	07	ETB	23	17			
BS	8	08	CAN	24	18			
HT	9	09	EM	25	19			
LF	10	0A	SUB	26	1A			
VT	11	0B	ESC	27	1B			
FF	12	0C	FS	28	1C			
CR	13	0D	GS	29	1D			
SO	14	0E	RS	30	1E			
SI	15	0F	US	31	1F			

11.18.13 SPECIAL CURSOR CODE STRINGS

Most of the cursor code strings are self-explanatory and consist of control characters, escape sequences, and other obvious codes.

11.18.14 COLUMN ONLY CURSOR POSITIONING

The "column only" cursor positioning is special because many terminals do not support this function. In terminals which do not support this function, there are three ways to simulate it. Terminals which do support "column only" positioning (e.g., ADDS), may use the terminal's normal control sequence (e.g., (CHAR(16) X). For terminals without "column only" positioning, the function may be simulated two ways. First, the cursor can be positioned to column zero of the current line (carriage return), followed by a cursor-right code for the number of columns required (e.g., CR STR(CHAR(12),X)). A variation of this is for VT-100 type terminals which may use a sequence like: CR ESC "[" X "C" BS, where the decimal value of X is part of the cursor-right escape sequence.

The other method of simulating "column only" positioning is less desirable, but may be effective in some instances. It uses the dummny cursor address variable Z in place of the Y address in a normal X-Y cursor address code (e.g., ESC "=" Z X).

11.18.15 CLEAR SCREEN & HOME @(-1)

The Clear Screen & Home code may consist of two different terminal control sequences (one for clear screen, and one for home). This is the case for VT-100 type terminals. Many other terminals combine these into one control sequence.

✓ PPG4 6/27/88

Chapter 12

PICK PC IMPLEMENTATIONS

THE PICK SYSTEM

USER MANUAL

Contents

12 PICK PC IMPLEMENTATIONS

12.1 INTRODUCTION 12-3
12.2 USING YOUR SYSTEM 12-4
12.2.1 Booting Your System 12-4
12.2.2 Shutting Down Your System 12-4
12.3 INSTALLING YOUR PICK SYSTEM 12-5
12.3.1 Preparations 12-5
12.3.2 New Installation 12-6
12.3.3 Upgrading Your PICK PC System 12-8
12.3.4 Full System Restore 12-8
12.3.5 File Restore from TCL 12-9
12.3.6 ABS Restore 12-10
12.3.7 Deleting the Partition 12-10
12.3.8 Configuration Extension 12-11
12.3.8.1 ABS Extension 12-11
12.3.8.2 Overlapped I/O 12-12
12.4 FIXED DISK PARTITIONING : FDISK 12-13
12.5 OFF-LINE STORAGE : STREAMING CARTRIDGE TAPE AND DISKETTES 12-16
12.5.1 Streaming Cartridge Tape (SCT) Commands 12-16
12.5.2 Diskette Commands 12-17
12.5.3 Additional Verbs 12-18
12.5.4 Examples 12-19
12.6 SETTING THE DATE FORMAT 12-20
12.7 MEMORY-MAPPED MONITOR 12-21
12.7.1 Keyboard Changes 12-21
12.7.2 Specifying Color and Display Modes 12-21
12.7.3 Keyboard Definition : SET-KBRD 12-23
12.7.4 Defining Function Keys : SET-FUNC 12-23
12.8 SERIAL PORTS 12-24
12.8.1 Data Carrier Detection : DCD 12-24
12.8.2 Flow Control : FC, MODEM 12-25
12.8.3 Setting Baud-Rate : SET-BAUD 12-25
12.8.4 Setting Port Characteristics : SET-PORT 12-26
12.8.5 Type-Ahead Capability : TA 12-27
12.8.6 Extended Character Set : XCS 12-27
12.8.7 Displaying Serial Line Characteristics : LIST-PORTS . 12-28
12.9 CREATING OR MODIFYING TERM TYPES : DEFINE-TERMINAL . . . 12-29
12.9.1 Menu Options 12-29
12.9.2 Describing Terminal Entries 12-33
12.10 TEST-CURSOR 12-38
12.11 OPTIMIZING PARALLEL PRINTER PERFORMANCE 12-39
12.12 DOS TO PICK BRIDGE : COPYDOS 12-41
12.13 PICK TO DOS BRIDGE : COPYPICK 12-45

Table(s)

12-1 Cursor Address Codes 12-34
12-2 Values Generated by Cursor Codes 12-35

12.1 INTRODUCTION

The PICK Personal Computer (PC) implementation includes several features for using the PICK system with your PC. These features include

- installing your system
- defining peripheral storage devices
- defining memory-mapped monitor features
- defining serial port and terminal characteristics
- copying files between PICK and MS-DOS[1] systems

---

[1] MS-DOS is a registered trademark of Microsoft Corporation.

12.2 USING YOUR SYSTEM

Your PC system can have up to four operating systems on it. The one that has control of the machine (as long as there is no diskette in the floppy drive) is the one that is on the partition marked 'active' by the FDISK command. (If a diskette is in the floppy drive, the PC tries to boot using that diskette.)

When PICK is installed, the installation procedure marks the partition containing PICK as the active partition. Therefore, any time you boot your system, unless you change the FDISK parameters or you insert a diskette in the floppy drive, you will come up in PICK.

12.2.1 Booting Your System

To boot the system, either press <ALT-CTRL-DEL> or turn the machine off, then on. You come up in the operating system in the active partition. If you are currently in PICK, you can reboot your system by using the REBOOT command.

FORMAT:

```
REBOOT
```

REBOOT checks that all users are logged off, executes the POWER-OFF verb which flushes memory to disk, then causes the system to boot just as if <ALT-CTRL-DEL> had been pressed. REBOOT works only if all other users are logged off. If any users are still logged on, REBOOT displays their account ids and line numbers, then returns to TCL.

This verb only works from line 0.

12.2.2 Shutting Down Your System

Before turning off your system, you should execute the POWER-OFF verb to ensure that all write-required frames are flushed from memory to disk.

FORMAT:

```
POWER-OFF
```

POWER-OFF disables all users, flushes memory to disk, and puts the machine into a HALTed state from which powering off and then back on is the only recovery. POWER-OFF only functions if all other users are logged off. If they are not, you will be informed who is still logged on and returned to TCL.

The operating system automatically flushes memory buffers to disk whenever the system has been quiescent for two seconds. If this automatic flush has already taken place, the system can be powered down without having to type POWER-OFF. However, using the POWER-OFF verb is the recommended procedure.

This verb only works from line 0.

12.3 INSTALLING YOUR PICK SYSTEM

Enclosed with each PICK system are the following diskettes:

PICK R83 PC imp System #1
PICK R83 PC imp System #2
PICK R83 PC imp System #3 (only if using 5-1/4 inch diskettes)
PICK R83 PC imp Data Files #1

NOTE: The imp entry specifies the implementation of hardware; for example, 286 specifies the PICK system for computers based on the 80286 microprocessor.

These diskettes are used to install new systems and to restore components of existing systems when necessary. The diskettes can be used as follows:

- New installation
- Upgrade of existing PICK system
- Full system restore; this replaces the monitor level code, the operating system object code, and the system level data files
- ABS restore; this replaces the monitor level code and the operating system object code
- Deletion of the PICK R83 partition

The hardware requirements to install a PICK system include:

- Supported PC implementation with one or two hard disks
- Minimum of 512Kb RAM
- Either monochrome or color/graphics monitor
- A minimum of 2.5Mb contiguous space available on the hard disk for a three user system; each additional user requires an estimated minimum of 0.2Mb

  NOTE: These figures for disk space are estimated minimums. PICK uses the largest partition that is available.

12.3.1 Preparations

The PICK system requires at least 2.5Mb of contiguous space on the hard disk for a three user system. If the disk does not have that much space available, you will have to copy the information from the disk (back up the disk), delete the existing partitions, create new, smaller partitions, then reinstall the operating system and other information back to the disk.

PICK installs itself on the largest available partition; therefore non-PICK operating systems should be installed before any PICK system.

If both the R83 and OA versions of PICK are to be installed, install a non-PICK operating system in the middle of the hard disk; this creates two additional partitions. Then install one PICK; it takes the larger partition. Install the second PICK; it takes the remaining partition.

For information on creating partitions for non-PICK operating systems, see the FDISK documentation supplied with that operating system.

12.3.2 New Installation

The first time a PICK system is placed on the computer, the installation process checks the PICK partition on the hard disk for disk errors, then loads all required components of the PICK operating system. The following procedure describes the steps.

1. Insert the diskette labeled PICK R83 PC imp System #1. To start the installation, press <ALT-CTRL-DEL> or power up the machine. The partition is checked and a message similar to the following is displayed:

```
PICK's R83 Ver n.n (date)
Virgin Install - PICK's R83 Ver n.n (date)


%%%%
Drive initialization
n:bbbbbb    status
```

where

n	drive number
bbbbbb	number of first block in group of 32 being checked
status	blank unless a block in the group is bad; if any block is bad, the word BAD! is displayed and the drive and block point to the bad group. The check continues; the information about bad groups is for your information. Groups with bad blocks are not used by PICK.

2. After the disk is checked, the following prompt is displayed:

```
Insert PICK System diskette #2 then type 'C' to continue:
```

Place the diskette labeled PICK R83 PC imp System #2 in the drive, then press C. The information on the diskette is loaded into the PICK area on the hard disk.

3. If using 5-1/4 inch diskettes, the following prompt is displayed:

```
Insert PICK System diskette #3 then type 'C' to continue:
```

Place the diskette labeled PICK R83 PC imp System #3 in the drive, then press C. The information on the diskette is loaded into the PICK area on the hard disk, then the hardware configuration is displayed.

4. The following prompt is displayed:

```
FILE RESTORE device:
H)igh density floppy,S)tandard density floppy,Q)uarter inch
SCT,O)ld SCT -
```

If using PICK Data File #1 on 5-1/4 inch diskette, enter H. If using PICK Data File #1 on 3-1/2 inch diskette, enter S. If using another media, enter the appropriate letter. (There may be a slight delay before the character you entered is displayed; this is normal.)

NOTE: Old SCT refers to tapes created by 2.1 and previous versions of R83, or by the R option of SET-SCT.

5. The system displays the following:

```
Load PICK Data Files #1 then type 'C' to continue:
```

6. Insert the diskette labeled PICK R83 imp Data Files #1, then press C. The files used by the system are placed on the hard disk. As each file is written, its name is displayed, similar to the following:

```
SPOOLER STARTED

 SYSTEM nnnn,n
 BLOCK-CONVERT nnnn,n
  .
  .
```

7. The system performs a coldstart, then displays the following:

```
Linking workspace; one moment please ...
```

8. The system logon prompt is displayed. Two accounts, SYSPROG and TUTOR, are available. To log on to SYSPROG, enter SYSPROG. All system functions are available. For information about the TUTOR account, see the chapter PICK TUTORIAL.

9. For more information about logging on, see the section on LOGON in the TCL chapter.

12.3.3 Upgrading Your PICK PC System

Upgrading an existing PICK PC system to a new version of PICK can be done by saving existing accounts, deleting the existing partition, installing the new operating system, and then restoring the accounts. The following procedure describes these steps.

1. Back up the existing system using FILE-SAVE.

2. Insert the R83 diskette labeled PICK R83 PC imp System #1. To start the procedure, press <ALT-CTRL-DEL> or power up the machine.

3. The following prompt is displayed:

   ```
   OPTIONS:  K)ill, A)BS only, F)ile & ABS, Q)uick file & ABS -
   ```

   Enter K to delete the existing partition. The following prompt requesting verification is displayed:

   ```
   Are you sure? (Y or N)
   ```

   To verify that the partition is to be deleted, enter Y. To abort the procedure, enter N.

   NOTE: It is important to delete the old partition to ensure that the correct boot record is put on to the hard disk.

4. The following prompt is displayed:

   ```
   PICK partition deleted...
   Press <ALT-CTRL-DEL> to reboot.
   ```

   To continue, press <ALT-CTRL-DEL>.

5. The procedure is then exactly the same as for a new installation. The PICK partition is checked for disk errors, then the new operating system is loaded. To continue, start with step 2 under New Installation.

6. Restore user accounts using RESTORE-ACCOUNTS. This restores all accounts that are not currently on the system; for example, it does not restore SYSPROG. Alternatively, you can restore each account individually using ACCOUNT-RESTORE. In this case, do NOT restore SYSPROG.

12.3.4 Full System Restore

A full system restore is used to replace the monitor level code, the operating system object code (the ABS area), and system level data files. It can also be used to restore an existing system to a previous file save if, for example, the system has been corrupted in some way.

There are two full restore processes available. One, the F-level restore, rechecks the hard disk before restoring any data; the other, the Quick restore, restores the data without rechecking the disk.

The following diskettes are required for a system restore:

PICK R83 PC imp System #1
PICK R83 PC imp System #2
PICK R83 PC imp System #3 (only if using 5-1/4 inch diskettes)
PICK R83 PC imp Data Files #1 OR the latest file save tapes from the existing system

The following procedure outlines the steps.

1. Insert the diskette labeled PICK R83 PC imp System #1.

2. To start the procedure, press <ALT-CTRL-DEL> or power up the machine.

3. The following prompt is displayed:

   OPTIONS: K)ill, A)BS only, F)ile & ABS, Q)uick file & ABS -

   To recheck the disk before restoring the ABS and data files, enter F. To skip the rechecking, enter Q.

4. The procedure is then exactly the same as a new installation, starting with step 2. If restoring an existing system, use your latest file save in place of the original PICK R83 PC imp Data Files #1 diskette.

12.3.5 File Restore from TCL

A file restore is used to replace the system data files, usually those created by a previous file save. This procedure does NOT restore any of the operating system code.

A file restore can be started from TCL, using the PROC :FILES. The following procedure outlines the steps.

1. Set the peripheral storage device, using SET-FLOPPY or SET-SCT, as appropriate.

2. Insert the first reel of data.

3. Ensure that all users are logged off.

4. Enter the following at the TCL prompt (the colon is necessary):

   :FILES

5. The system is shutdown, then the files restored.

12.3.6 ABS Restore

An ABS restore replaces the monitor level code and the operating system object code (ABS area); it does not update any data files.

The following procedure outlines the steps.

1. Insert the diskette labeled PICK R83 PC imp System #1. To start the procedure, press <ALT-CTRL-DEL> or power up the machine.

2. The following prompt is displayed:

   OPTIONS: K)ill, A)BS only, F)ile & ABS, Q)uick file & ABS

   To restore the ABS, enter A.

3. The following prompt is displayed:

   Insert PICK System diskette #2 then type 'C' to continue:

   Place the diskette labeled PICK R83 PC imp System #2 in the drive, then press C.

4. The following prompt is displayed:

   Insert PICK System diskette #3 then type 'C' to continue:

   Place the diskette labeled PICK R83 PC imp System #3 in the drive, then press C. The ABS is loaded into the PICK area on the hard disk, then the hardware configuration is displayed.

5. The system performs a coldstart and the system logon prompt is displayed.

12.3.7 Deleting the Partition

To delete an existing PICK partition, use the following procedure:

1. Insert the diskette labeled PICK R83 PC imp System #1. To start the procedure, press <ALT-CTRL-DEL> or power up the machine.

2. The following prompt is displayed:

   OPTIONS: K)ill, A)BS only, F)ile & ABS, Q)uick file & ABS

   Enter K to delete the existing partition. The following prompt requesting verification is displayed:

   Are you sure? (Y or N)

   To verify that the partition is to be deleted, enter Y. To abort the procedure, enter N.

3. The following prompt is displayed:

```
PICK partition deleted...
Press <ALT-CTRL-DEL> to reboot.
```

To reboot using the PICK R83 PC imp System #1 diskette, press <ALT-CTRL-DEL>. To reboot using another operating system, insert the appropriate diskette. To reboot from the hard disk, remove all diskettes.

12.3.8 Configuration Extension

During an initial virgin install procedure, a proprietary message is displayed. Following this proprietary message, there is a three-second window during which the install process can be interrupted in order to change the configuration. The window is noted by the display of percent signs (%), about one per second. The window is closed when the fourth percent sign is displayed.

Currently, there are two configuration parameters that can be changed:

- The size of the ABS area may be increased to accommodate additional assembly level requirements. This extension should not be done unless absolutely necessary. Needless ABS extension wastes available disk space.

- Streaming Cartridge Tape (SCT) and disk I/O can be toggled between overlapped and non-overlapped processing. Overlapped processing is dependent on the hardware; the default is non-overlapped processing.

After the configuration change has been made, the system then restarts the display of percent signs. You may re-enter either an A or S.

12.3.8.1 ABS Extension

ABS should be extended only in the following cases:

- Your application is composed of assembler programs that exceed a total of 100 frames.

- Your assembler program development is anticipated to require more than 100 frames.

To extend your ABS, press the letter A before the fourth per cent sign is displayed. After A is pressed, the following is displayed:

```
Enter total ABS frames (704 <= #ABS <= 4096) =
```

Enter the total number of frames desired. The number of ABS frames is increased in increments of 32 frames; if necessary, the system rounds your number up to the next multiple of 32.

CAUTION: The ABS area must reside totally on the first hard disk. (4096 ABS frames use 8Mb of disk.)

NOTE: If you have extended your ABS and need to call technical support for any reason, be sure to indicate that you have an 'ABS EXTENDED SYSTEM'.

12.3.8.2 Overlapped I/O

Overlapped I/O improves the performance of your system when you are using streaming cartridge tape; however, your hardware must be able to support the overlapping.

To change the configuration for overlapped I/O, press the letter S before the fourth percent sign is displayed. The system toggles the overlap, then displays its current status, similar to the following:

```
Streaming Tape I/O and Hard Disk I/O will (not) be overlapped.
```

The actual message displays not, as appropriate.

12.4 FIXED DISK PARTITIONING : FDISK

The Fixed Disk Partitioning concept allows up to four different operating systems to co-reside on one or two drives. The verb FDISK has been provided to support the use of the PC's Fixed Disk Partitioning concept. It can be run only from the SYSPROG account.

FORMAT:

```
>FDISK
```

With fixed disk partitions, each operating system resides on the hard disk within its own range of contiguous cylinders. The FDISK command allows the user control over which operating system is currently executing by marking one of the co-resident operating systems as "active". When the system is booted, the operating system marked as "active" assumes control of the machine. Therefore, to move from operating system A to operating system B, invoke FDISK, mark B as "active", then reboot the system. B is now the active operating system.

The PICK system FDISK is similar to the FDISK implemented under MS-DOS. It differs from the MS-DOS version as follows:

- The option to create a partition is not functional under PICK because partition creation is handled automatically at system installation time.

- The delete partition option refuses to delete the PICK partition unless another partition is first marked "active". In the case where PICK is the only operating system, this does not apply.

- Deleting the PICK partition from drive C also deletes PICK from drive D, if it exists. With the PICK FDISK, the user may view but not modify the partitions on drive D.

FDISK can be used to display the current fixed disk partition parameters and current disk configuration, or to change the active partition. This information is kept on the Master Fixed Disk Boot Record (MFDBR); when the system is booted, the operating system on the partition marked active in the MFDBR assumes control of the computer. If no partition is marked active, the system cannot be booted from hard disk.

When FDISK is invoked, a screen similar to the following is displayed:

```
The PICK System
Fixed Disk Setup Program
(c)Copyright PICK Systems 1986, 1987, 1988

FDISK Options

Current fixed disk drive: 1

Choose one of the following:

    0.  Quit & Return to TCL
    1.  Create PICK Partition
    2.  Change Active Partition
    3.  Delete PICK Partition
    4.  Display Partition Data
    5.  Select Next Fixed Disk Drive
    6.  Undo Changes to Partition Data

Enter choice: [ ]
```

To exit FDISK and return to the TCL prompt, enter 0.

Option 1, Create PICK Partition, displays the current partitions defined in the MFDBR, but does not actually do anything else; it is provided for compatibility with the MS-DOS FDISK command.

Option 2, Change Active Partition, is used to select the partition that contains the desired operating system. The partition that is marked active assumes control the next time the system is booted.

Option 3, Delete PICK Partition, is used to delete from the MFDBR the reference to the partition that currently has control of the computer (the currently active partition). If there is more than one partition, then before the currently active partition can be deleted, another partition must be marked active using option 2. The only partition that can be deleted is the currently active partition.

Option 4, Display Partition Data, is used to display the status of all partitions, and is similar to the following:

```
PARTITION   STATUS   TYPE   START    END  CYLNDRS  SIZE MB
    1         N      DOS    0000   0255    0256    31.88
    2         A      R83    0256   0898    0643    80.06

Total fixed disk space is 899 Cylinders, 111.94 Mbytes

Press ESC to return to FDISK option [ ]
```

Option 5, Select Next Fixed Disk, displays the statistics for the current drive, and if the system supports two drives, gives the option to select one drive or the other. The statistics that are displayed are similar to the following:

```
Fixed disk drive is 1 of 1

     Heads.....    15
     Cylinders....  899
     Sectors......229245
     Mega bytes...111.94

     Press ESC to return to FDISK option [ ]
```

Option 6, Undo Changes to Partition Data, returns the partition settings in the MFDBR to those that were in effect when FDISK was invoked.

12.5 OFF-LINE STORAGE : STREAMING CARTRIDGE TAPE AND DISKETTES

The PICK PC System supports 1/4" streaming cartridge tape (SCT) and both 5-1/4 inch and 3-1/2 inch diskettes as peripheral storage devices. The desired device is assigned to your line with a SET-SCT or a SET-FLOPPY command. (SET-FLOPPY is used for both sizes of diskettes.)

For information on the installation of peripheral storage devices, see the PC PERIPHERAL INSTALLATION GUIDE.

12.5.1 Streaming Cartridge Tape (SCT) Commands

The SET-SCT command assigns the peripheral storage device to the SCT unit. The SET-SCT command also does an automatic tape attach (T-ATT) of the SCT unit to your line.

FORMAT:

```
SET-SCT {buffers} {({blk-size}{R})
```

where

buffers — number of buffers to assign to SCT processing; each buffer is 512 bytes (1/2Kb). The default (and minimum number) is 128. The maximum number of buffers depends on the size of main memory for your system; up to one half your memory can be reserved for buffers. For example, if you have a 512Kb system, you can have up to 512 buffers.

NOTE: If you are using SCT while other processes are active, using the maximum number of buffers may adversely affect the performance of those processes.

blk-size — size of one block on the SCT; may be any value between 500 and 32500. For maximum efficiency, it is recommended that the block size be a multiple of 512. The default block size is 16384.

R — use the old SCT format; the old format writes a 512-byte block between each block of data (the current SCT format is a continuous stream of blocks of the specified size). The old format is used by 2.1 and previous versions of R83. If the R option is specified, the block size must be a multiple of 512 and must be in the range 2560 and 16384.

The T-RETEN command retensions the SCT by first forwarding, then backspacing the tape its entire length (no data is destroyed). It is recommended that tapes be retensioned before they are used to reduce the occurrence of parity errors.

FORMAT:

```
T-RETEN
```

The T-ERASE command also retensions the SCT by first forwarding, then backspacing the tape its entire length; however, T-ERASE also erases the entire tape.

FORMAT:

T-ERASE

12.5.2 Diskette Commands

The SET-FLOPPY command sets the peripheral storage device to a diskette drive. The SET-FLOPPY command also does an automatic tape attach (T-ATT) of the diskette drive to your line.

FORMAT:

SET-FLOPPY {(density,drive)

The options for density are

H High Density; 1.2 Mb for 5-1/4 inch drives, 1.44 Mb for 3-1/2 inch drives

S Standard Density; 360Kb for 5-1/4 inch drives, 720Kb for 3-1/2 inch drives

The options for drive are

A Drive A:

B Drive B:

If no options are specified, the drive is set to drive A as the default and the density is set to the highest density supported by the drive.

Diskettes must be formatted for PICK before they can be used. To format diskettes, use the verb FORMAT. The density and drive specified by the last SET-FLOPPY verb are used as the defaults. If any bad sectors are detected by the format process, the diskette is not usable under PICK.

FORMAT:

FORMAT

After the verb has been entered, a screen similar to the following is displayed:

```
BLOCK SIZE 500
PICK System Floppy Disk Format Utility

Format (size) diskette media
Insert diskette in floppy drive (drive)
and strike any key when ready
```

After the formatting has been completed, statistics similar to the following are printed:

Good format:

```
Formatting
++++++++++++++++ . . .
completed.

   nn bytes total disk space
   mm bytes available on disk

Format another (Y/N) -
```

Bad format:

```
Formatting:
+++-+++++-+---+++++- . . .
completed.

   nn bytes total disk space
   mm bytes available on disk
   xx bytes in bad sectors
   This diskette is unusable by PICK

Format another (Y/N) -
```

CAUTION! PICK and MS-DOS use different formats for disks. A disk that is formatted for PICK is not usable by MS-DOS.

12.5.3 Additional Verbs

The following verbs can be used with both SCT and diskettes:

T-STATUS

This command returns with drive and type information of the currently assigned peripheral storage device.

T-REW

This command repositions the peripheral storage device to the beginning of the media. In addition, after write operations using SCT, the T-REW writes a terminating EOF mark before rewinding the tape.

You should always perform a T-REW before removing the media from the drive.

12.5.4 **Examples**

The following examples illustrate the use of the peripheral storage device commands.

SET-SCT	Set tape device to 1/4" tape, use default block size.
SET-FLOPPY (SB	Set tape device to floppy drive B, standard density
SET-FLOPPY (HA	Set tape device to floppy drive A, high density.
SET-FLOPPY	Set tape device to floppy drive A, highest density supported by that drive

Sample usage of SET-SCT and SET-FLOPPY commands.

12.6 SETTING THE DATE FORMAT

Both European date format and USA data format are supported on the PICK PC system.

FORMAT:

SET-DATE-EUR

SET-DATE-EUR specifies that the numeric date format is dd/mm/yy. The default display remains dd mon yyyy.

FORMAT:

SET-DATE-STD

SET-DATE-STD specifies that the numeric date format is mm/dd/yy. The default date format remains dd mon yyyy.

For information on setting the date, see the SET-DATE verb in Chapter 3, Terminal Control Language.

Statement	Description
SET-DATE-EUR	Sets the date to European format; if a numeric format is requested, the date is displayed as 30*9*88, where * is the specified delimiter.
SET-DATE-STD	Sets the date to the USA format; if a numeric display is requested, the date is displayed as 9/30/88, where / is the specified delimiter. The default display remains 30 Sep 1988.

12.7 MEMORY-MAPPED MONITOR

On PICK PC systems, the device on line 0 is always assumed to be a memory-mapped monitor. PICK includes verbs to set color, keyboards, and function keys on memory-mapped monitors.

12.7.1 Keyboard Changes

The PC memory-mapped monitor keyboard has certain features that are different from standard CRT capabilities. These features have been disabled under the PICK Operating System to make the monitor appear to be a standard CRT keyboard.

The following is a list of changes under PICK:

- Keypad area generates numerics only.
- Backtabs cannot be generated from the keyboard.
- Print screen functions are disabled.
- The function keys F1 to F10 are user-definable via the included BASIC program SET-FUNC.
- The <ALT> key recognition is disabled except in the following cases:

<ALT-CTRL-DEL>	reset sequence
<ALT-CTRL-BREAK>	system debugger sequence
<ALT-n>	character displayed on the vertical face of the key
<ALT-functionkey>	user-defined function key

12.7.2 Specifying Color and Display Modes

Two verbs are available from TCL that allow you to specify the color and mode of the monitor on line 0: COLOR and MONO.

FORMAT:

COLOR {foreground color}{,background color}{modes}

The following colors are available for background and foreground:

BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, WHITE

The following modes are available:

/B or /BLINK	Activate character blinking
/NB or /NOBLINK	De-activate character blinking
/F or /FULL	Full intensity foreground

/H or /HALF	Half intensity foregound
/R or /REVERSE	Activate reverse video
/NR or /NOREVERSE	De-activate reverse video

STATEMENT	EXPLANATION
COLOR RED	Set foreground to red.
COLOR ,BLUE	Set background color to blue.
COLOR /B	Activate character blinking.
COLOR /NB/R	Deactivate character blinking, activate reverse video.
COLOR BROWN/F	Set foreground to brown and full intensity.
COLOR ,RED/H	Set background to red, set foreground to half-intensity.
COLOR GREEN,CYAN/HALF	Set foreground to half intensity green, background cyan.

Sample usage of the COLOR verb.

The COLOR verb works only on line 0 and only if a COLOR/GRAPHICS adapter is on the system.

FORMAT:

MONO {modes}

The supported switches are identical to the COLOR verb switches, with the addition of the following:

/U or /UNDERLINE	Activate character underlining
/NU or /NOUNDERLINE	Deactivate character underlining

The MONO verb works only on line 0 and only if a monochrome adapter card is in the system.

12.7.3 Keyboard Definition : SET-KBRD

The SET-KBRD program is used to redefine the keyboard for memory-mapped monitors. SET-KBRD applies only to line 0.

FORMAT:

```
>SET-KBRD file.name item.name
```

where

filename name of the file that contains keyboard definitions

item.name name of specific keyboard item

Several keyboard definition items are included with the system. These may be found in the file KEYBOARDS in the account SYSPROG and include:

BRITISH
FRENCH
GERMAN
ITALIAN
SPANISH
USA

The default keyboard is the USA keyboard. For more information about keyboard definition items, see Appendix A, Creating Keyboard Definition Items.

12.7.4 Defining Function Keys : SET-FUNC

The SET-FUNC program is used to redefine the function keys for memory-mapped monitors. SET-FUNC applies only to line 0.

FORMAT:

```
>SET-FUNC file.name item.name
```

where

filename name of file that contains function key definitions

item.name name of specific function key item

The function keys are initially defined by the default keyboard definition item. SET-FUNC overrides the default settings or a previously executed SET-KBRD; however, if SET-KBRD is executed after SET-FUNC is executed, the function keys are reset according to the keyboard definition item used.

Some example function key definition items are included with the system. These may be found in the file FUNCKEYS on the SYSPROG account. For more information about function key definitions, see Appendix B, Creating Function Key Items.

12.8 SERIAL PORTS

Several commands are available that allow you to set up characteristics for devices running on the serial ports of your PC. The commands that are available include the following:

DCD	Data Carrier Detection
FC	Flow Control
MODEM	Flow Control
SET-BAUD	Baud Rate
SET-PORT	Baud Rate, Parity, Stop Bits, Word Length
TA	Type Ahead
XCS	Extended Character Set

A command, LIST-PORTS, can be used to display the characteristics of all configured ports.

12.8.1 Data Carrier Detection : DCD

The DCD protocol allows the monitor to sense changes in the DCD signal. When the monitor senses a loss of the DCD signal on a line, it logs the line off. When the system is booted, the DCD protocol is deactivated.

FORMAT:

```
DCD {line.number}
DCD-ON {line.number}
DCD-OFF {line.number}
```

DCD can be used to display the current status of DCD protocol on a line; DCD-ON can be used to activate the data carrier detect (DCD) protocol for a specified line; DCD-OFF can be used to deactivate the data carrier detect (DCD) protocol.

If the line number is not specified, the current line is assumed.

NOTE: The cabling for the device must allow the DCD signal to reach the system. See the cabling instructions for modems in the PICK PERIPHERAL INSTALLATION GUIDE.

12.8.2 Flow Control : FC, MODEM

Flow control protocol allows the system to monitor the DSR (Data Set Ready) signal and the software X-ON/X-OFF protocol. When DSR signal is off or the X-OFF protocol is invoked, the output to the serial port is suspended.

When the system is booted, flow control is on for all ports.

FORMAT:

```
FC {line.number}
FC-ON {line.number}
FC-OFF {line.number}
```

FC can be used to display the current status of flow control on a line; FC-ON can be used to enable the flow control for a specified line; FC-OFF can be used to disable the flow control.

If the line number is not specified, the current line is assumed.

Flow control can also be enabled or disabled using the MODEM commands.

FORMAT:

```
MODEM-ON
MODEM-OFF
```

12.8.3 Setting Baud-Rate : SET-BAUD

The SET-BAUD verb allows serial ports to be set to various baud rates.

FORMAT:

```
SET-BAUD {line.number,}baud-rate
```

The SET-BAUD verb effects only serial ports. If the line number parameter is not present, the baud rate is set for the line to which you are currently logged.

The following baud rates are supported:

110	2400
150	4800
300	9600
600	19200
1200	

STATEMENT	EXPLANATION
SET-BAUD 1,4800	Sets port 1 to 4800 baud.
SET-BAUD 9600	Sets the current port to 9600 baud.

Sample usage of the SET-BAUD verb.

12.8.4 Setting Port Characteristics : SET-PORT

SET-PORT is used to display or define the baud rate, parity, stop bits, and data length for a specified line (port).

FORMAT:

SET-PORT line.number{,baud,parity,stop.bits,word.len}

where

line.number — serial line on which to change setup; if no line is specified, the current line is assumed. If the line number is not a valid line number for the current system, the following message is displayed:

ILLEGAL LINE NUMBER

baud — the following baud rates are supported:

110	2400
150	4800
300	9600
600	19200
1200	

No other values are accepted. If other values are entered, an error message is displayed.

parity — the following values are supported:

N{ONE} no parity
O{DD} odd parity
E{VEN} even parity

stop.bit — number of stop bits; may be 1 or 2

word.len — data length; may be 7 or 8

If no options are entered, the current port settings are displayed.

If an option is not changing, just a comma may be entered.

Statement	Description
>SET-PORT 6,,N,1,8	Sets the line for the terminal on line 6. The baud rate is not changing, so just a comma was entered.
Line number : 6	
Baud rate : 9600	
Parity : NONE	
Stop bits : 1	
Word length : 8	

12.8.5 Type-Ahead Capability : TA

Type-ahead is enabled for all lines when the system is booted. TA can be used to display the current status of type-ahead; TA-On can be used to turn on type-ahead; TA-OFF can be is used to turn off type-ahead.

FORMAT:

```
TA {line.number}
TA-ON {line.number}
TA-OFF {line.number}
```

If no line number is specified, the current line is assumed.

NOTE: This command is also valid for line 0.

12.8.6 Extended Character Set : XCS

XCS is used to display or turn on or off the extended character set capability of serial devices on a specified line.

FORMAT:

```
XCS {line.number}
XCS-OFF {line.number}
XCS-ON {line.number}
```

where
line.number line to affect

When XCS is on, the high order bit is not stripped from input that is transmitted to the system, thus allowing characters in the range 128-239 to be used as unique characters. When XCS is off, the high order bit is stripped from input, and characters 128-239 are not available.

When the system is booted, the extended character set capability is off.

Statement	Description
>XCS	Displays current setting of current line.
>XCS-ON 6	Turns on the extended character set capability on line 6.

12.8.7 Displaying Serial Line Characteristics : LIST-PORTS

The current settings of the serial ports on the system can be displayed using the command LIST-PORTS.

FORMAT:

```
LIST-PORTS {(Z)
```

The Z option displays the characteristics for all configured lines. If Z is not specified, only the lines currently connected are displayed.

The following characteristics are displayed:

- Line number
- Baud rate
- Parity
- Stop bits
- Word length
- Type-ahead
- Flow control
- Carrier detect
- Extended character set

12.9 CREATING OR MODIFYING TERM TYPES : DEFINE-TERMINAL

The DEFINE-TERMINAL utility can be used to create or modify term types.

FORMAT:

```
>DEFINE-TERMINAL
```

The DEFINE-TERMINAL utility can be used to customize the terminal characteristics functions for your system. The utility is a menu-driven BASIC program which creates, maintains, and compiles terminal definitions that are then kept in the CURSOR file on the SYSPROG account. In addition, the utility can be used to select up to 26 different terminals from the CURSOR file for inclusion in the System Cursor table of active terminal definitions for your system. Although only 26 terminals may be selected for inclusion in this table, any number of terminals may be defined by this utility.

When DEFINE-TERMINAL is invoked, a menu similar to the following is displayed:

```
System Cursor Definition Utility

The following terminals are defined.  Terminals marked with an
asterisk (*) are selected to be included in your System Cursor
Definition.

*A ADDS            H HONEYWELL     *N WYSE100      *V VIEWPOINT
 B BEEHIVE         I IBM3010       *Q MIME         *W WYSE50
 C DTC            *J VT100         *R REGENT        X DATAGRAPHIX
 D DATAMEDIA      *K VT52          *S SOROC
 E EMULOG200      *L LSI            T TEC
 G GTC            *M AMPEX         *T TV920

     1)  Create Terminal Definition
     2)  Modify Terminal Definition
     3)  Delete Terminal Definition
     4)  Add Terminal to Selected Definitions
     5)  Delete Terminal from Selected Definitions
     EX  Exit without updating System Cursor
     FI  Update System Cursor to selected terminals

         Enter Selection (1-5) or EX or FI:
```

12.9.1 Menu Options

Each of the menu options is explained in the following sections.

CREATE TERMINAL DEFINITION

Option 1 allows the creation of a new terminal definition. A terminal definition consists of a series of parameters which the system requires to control a particular terminal (type, size, control codes, etc.).

After you enter the menu selection 1, the routine prompts for the terminal name to be defined. If the name already exists, you may opt to modify the existing definition or enter another name. If you opt to modify the existing definition, then the routine will proceed as in the next section (modification).

If the name is new, you will be asked if you want to use a copy of an existing terminal definition for the initial values for the new definition. If so, you will be prompted for the name of the existing terminal to be used as a "template". This is useful for defining terminals which are similar to other existing terminals. If you do not choose to use an existing terminal definition as a "template", then the routine proceeds to prompt for each of the parameters for the new definition.

MODIFY TERMINAL DEFINITION

Option 2 allows the modification of existing terminal definitions. After you enter the menu selection 2, the routine prompts for the name of the terminal definition to be modified. If the name does not exist, you may opt to create it. If you opt to create a new definition, then the routine proceeds as in the previous section (new definition). Otherwise, the routine proceeds to the definition modification mode.

In the terminal definition modification mode, the set of parameters for the terminal is broken into page size blocks for display and modification. First, a section of the existing definition is displayed, then the prompt "Modify Lines?" is issued. You may enter N (default), Y, list of line numbers, or a range of line numbers to modify.

If you enter N, the next section of the existing definition is displayed and the process repeats until the entire definition has been reviewed.

If you enter Y, you are prompted for the line numbers.

If you enter a list of line numbers or a range of line numbers, you are prompted to enter new data for each of the specified lines. At each of these prompts, the following entries may be made:

- new parameters; replaces existing values
- a carriage return (null value); causes the data for the line to be unchanged.
- any number of spaces; causes the data for the line to be changed to null.
- a single questions mark (?); presents a brief explanation of the contents of the line, and then reprompts for input.

After all the lines have been prompted for, the section and Modify Lines? prompt are redisplayed.

Once all the sections have been reviewed, you will be asked if the terminal definition is correct. If not, the review and modify process will be repeated, or you may exit without saving any modifications.

If the definition is correct, an attempt will be made to "compile" it. If the compilation detects errors, you can correct the errors via the modification process. Once the definition is compiled, you can select the terminal to be included in the list of terminals for your System Cursor table.

The following is an example of the display portion of the modification mode:

```
Terminal - TV920

   1.  TYPE.............................. T
   2.  SCREEN SIZE....................... 80,24
   3.  CURSOR ADDRESS CODE............... L
   4.  @(X) CURSOR POSITIONING........... CR STR(CHAR(12),X)
   5.  @(X,Y) CURSOR ADDRESSING.......... ESC "=" Y X
   6.  @(-1) CLEAR SCREEN & HOME......... CHAR(26)
   7.  @(-2) CURSOR HOME................. CHAR(30)
   8.  @(-3) CLEAR TO END OF PAGE........ ESC "Y"
   9.  @(-4) CLEAR TO END OF LINE........ ESC "T"
  10.  @(-5) START BLINK................. ESC "^"
  11.  @(-6) STOP BLINK.................. ESC "q"
  12.  @(-7) START PROTECT................ ESC ")"
  13.  @(-8) STOP PROTECT................. ESC "("
  14.  @(-9) CURSOR BACK.................. BS
  15.  @(-10) CURSOR UP................... VT
Modify lines? NO
Terminal - TV920

  16.  @(-11) ENABLE PROTECT MODE ....... ESC "&"
  17.  @(-12) DISABLE PROTECT MODE ...... ESC HEX(27)
  18.  @(-13) START REVERSE VIDEO........ ESC "j"
  19.  @(-14) STOP REVERSE VIDEO......... ESC "k"
  20.  @(-15) START UNDERLINE............ ESC "l"
  21.  @(-16) STOP UNDERLINE............. ESC "m"
  22.  @(-17) SLAVE ON ..................
  23.  @(-18) SLAVE OFF ..................
  24.  @(-19) CURSOR FORWARD ............. FF
  25.  @(-20) CURSOR DOWN ................ LF
  26.  @(-21) GRAPHICS CHARACTER SET ON ....
  27.  @(-22) GRAPHICS CHARACTER SET OFF ...
  28.  @(-23) KEYBOARD LOCK ..............
  29.  @(-24) KEYBOARD UNLOCK ............
  30.  @(-25) CONTROL CHARACTER ENABLE .....
Modify lines? NO
```

```
Terminal: TV920

  31.  @(-26) CONTROL CHARACTER DISABLE ....
  32.  @(-27) WRITE STATUS LINE ............
  33.  @(-28) ERASE STATUS LINE ............
  34.  @(-29) INITIALIZE TERMINAL MODE .....
  35.  @(-30) DOWNLOAD FUNCTION KEYS .......
  36.  @(-31) NON-EMBEDDED STAND-OUT ON ....
  37.  @(-32) NON-EMBEDDED STAND-OUT OFF ...
  38.  @(-99) EMBEDDED VISUAL ATTRIBUTES?...
  39. @(-100) HALF INTENSITY ...............
  40. @(-101) FULL INTENSITY ...............

Modify lines? NO
Is table for terminal TV920 correct? YES
```

DELETE TERMINAL DEFINITION

Option 3 allows for the deletion of terminal definitions. After entering the menu choice 3, you are prompted for the name of the terminal to be deleted.

ADD TERMINAL TO SELECTED DEFINITIONS

Option 4 allows for the addition of a terminal to the list of terminals to be included in your System Cursor table. After entering the menu selection 4, you are prompted for the name of the terminal to be added to the list of selected terminals. If the name exists, then the routine checks if that type of terminal (terminal type in the definition) has already been selected. If not, the desired terminal is added to the selected definitions in the System Cursor table.

If the type of the desired terminal has already been selected for another terminal, you will be asked if you want to replace the previous selection with the new selection. If so, the previous selection will be deleted from the list of selected terminals, and the new selection added.

DELETE TERMINAL FROM SELECTED DEFINITIONS

Option 5 allows for the deletion of a terminal from the list of terminals to be included in your System Cursor table.

EXIT WITHOUT UPDATING SYSTEM CURSOR

Option EX quits the definition process without updating the operating System Cursor table. All the modifications and selections made are preserved.

UPDATE SYSTEM CURSOR TO SELECTED TERMINALS

Option FI quits the definition process after updating the operating System Cursor table with the new selections.

12.9.2 Describing Terminal Entries

Some of the entries used in the DEFINE-TERMINAL utility are described here.

TYPE

The terminal type is a single upper-case letter which identifies the terminal to the system. The terminal type field in the terminal table corresponds to the type as set with the TERM command.

SCREEN SIZE

The size field contains the screen size in columns and rows. The size is entered as two numbers separated by a comma (e.g., 80,24). If a value exceeds the size, then the maximum size is substituted.

CURSOR ADDRESS CODE

The cursor address code is usually a single letter (A, L, T, H, D). The defined types are "A" for ADDS type addressing, "L" for Lear-Seigler type addressing, "T" for TEC type addressing, "H" for Hazeltine type addressing and "D" for decimal type addressing.

All types except "D" produce binary column and row addresses (single byte for each). "D" type addressing produces one to three digits for column and row addresses. If "D" type addressing is used, the code may be followed by two digits (22, 23, 32, 33) to force padding to the desired number of digits (e.g., "D32" will produce decimal addressing with 3 digits used for the column and 2 digits for the row (leading zeros added to force the length). "D" alone will use "floating" decimal numbers from 1 to 3 digits.

All cursor addressing codes may be followed by a plus sign "+" which adds one to the column and row addresses before generating the address codes. This allows for terminals which define the upper-left corner of the screen as "1,1" instead of "0,0". Thus, decimal addressing with a three digit row and column address numbered from "1,1" would be: "D33+".

The following table displays the algorithm each of the cursor address codes uses to position the cursor.

Table 12-1 Cursor Address Codes

CODE	Algorithm
A	COL = CHAR((INT(X/10)*6)+X) ROW = CHAR(Y+64)
L	COL = CHAR(X+32) ROW = CHAR(Y+32)
T	COL = CHAR(-(1+X)) ROW = CHAR(-(1+Y))
H	COL = CHAR(X) ROW = CHAR(Y)

The following table shows the column or row address, and the associated code needed to position the cursor.

Table 12-2 Values Generated by Cursor Codes

X	Y	A COL	A ROW	L	T	H	X	Y	A COL	A ROW	L	T	H
0	0	nul	@	space	del	nul	40		@		H	W	(
1	1	soh	A	!	~	soh	41		A		I	V	)
2	2	stx	B	"	}	stx	42		B		J	U	*
3	3	etx	C	#	|	etx	43		C		K	T	+
4	4	eot	D	$	{	eot	44		D		L	S	,
5	5	eng	E	%	z	eng	45		E		M	R	-
6	6	ack	F	&	y	ack	46		F		N	Q	.
7	7	bel	G	'	x	bel	47		G		O	P	/
8	8	bs	H	(	w	bs	48		H		P	O	0
9	9	ht	I	)	v	ht	49		I		Q	N	1
10	10	dle	J	*	u	lf	50		P		R	M	2
11	11	dc1	K	+	t	vt	51		Q		S	L	3
12	12	dc2	L	,	s	ff	52		R		T	K	4
13	13	dc3	M	-	r	cr	53		S		U	J	5
14	14	dc4	N	.	q	so	54		T		V	I	6
15	15	nak	O	/	p	si	55		U		W	H	7
16	16	syn	P	0	o	dle	56		V		X	G	8
17	17	etb	Q	1	n	dc1	57		W		Y	F	9
18	18	can	R	2	m	dc2	58		X		Z	E	:
19	19	em	S	3	l	dc3	59		Y		[	D	;
20	20	space	T	4	k	dc4	60		'		\	C	<
21	21	!	U	5	j	nak	61		a		]	B	=
22	22	"	V	6	i	syn	62		b		^	A	>
23	23	#	W	7	h	etb	63		c		_	@	?
24		$		8	g	can	64		d		'	?	@
25		%		9	f	em	65		e		a	>	A
26		&		:	e	sub	66		f		b	=	B
27		'		;	d	esc	67		g		c	<	C
28		(		<	c	fs	68		h		d	;	D
29		)		=	b	gs	69		i		e	:	E
30		0		>	a	rs	70		p		f	9	F
31		1		?	'	us	71		q		g	8	G
32		2		@	_	space	72		r		h	7	H
33		3		A	^	!	73		s		i	6	I
34		4		B	]	"	74		t		j	5	J
35		5		C	\	#	75		u		k	4	K
36		6		D	[	$	76		v		l	3	L
37		7		E	Z	%	77		w		m	2	M
38		8		F	Y	&	78		x		n	1	N
39		9		G	X	'	79		y		o	0	O

CURSOR CODE STRINGS

The cursor code strings are expressions which produce the control and escape sequences used by the terminal being defined. The expressions are similar to BASIC syntax, except that a blank may be used between elements in the expression as well as a colon. Cursor code strings may consist of the following separated by blanks or colons:

1) Defined control character (e.g., ESC, BS, DEL, etc.)
2) String literal in quotes (e.g., "A", '[0', etc.)
3) Character function (e.g., CHAR(21))
4) Hexadecimal string (e.g., HEX(1B41))
5) String function (e.g., STR(NUL,5) or STR(CHAR(12),X00
6) Cursor address variable (e.g., X, Y, or Z)

The cursor address variables (X, Y, Z) cause the specified address (byte or decimal string) to be inserted into the control string at the specified position. The variable X contains the column, Y contains the row, and Z contains the row previously referenced in an @(X,Y) code (or zero if the last reference was @(-1) or @(-2).

The symbolic name for the control codes and their decimal and hexadecimal equivalents are shown in the table below. Any of these codes may be included in the cursor code string. It is often easier to reference the backspace character as BS instead of CHAR(8), or NUL instead of CHAR(0).

CODE	DEC	HEX	CODE	DEC	HEX	CODE	DEC	HEX
NUL	0	00	DLE	16	10	SP	32	21
SOH	1	01	DC1	17	11	DEL	127	22
STX	2	02	DC2	18	12			
ETX	3	03	DC3	19	13			
EOT	4	04	DC4	20	14			
ENQ	5	05	NAK	21	15			
ACK	6	07	SYN	22	16			
BEL	7	07	ETB	23	17			
BS	8	08	CAN	24	18			
HT	9	09	EM	25	19			
LF	10	0A	SUB	26	1A			
VT	11	0B	ESC	27	1B			
FF	12	0C	FS	28	1C			
CR	13	0D	GS	29	1D			
SO	14	0E	RS	30	1E			
SI	15	0F	US	31	1F			

SPECIAL CURSOR CODE STRINGS

Most of the cursor code strings are self-explanatory and consist of control characters, escape sequences, and other obvious codes.

COLUMN ONLY CURSOR POSITIONING

The "column only" cursor positioning is special because many terminals do not support this function. In terminals which do not support this function, there are three ways to simulate it. Terminals which do support "column only" positioning (e.g., ADDS), may use the terminal's normal control sequence (e.g., (CHAR(16) X). For terminals without "column only" positioning, the function may be simulated two ways. First, the cursor can be positioned to column zero of the current line (carriage return), followed by a cursor-right code for the number of columns required (e.g., CR STR(CHAR(12),X)). A variation of this is for VT-100 type terminals which may use a sequence like: CR ESC "[" X "C" BS, where the decimal value of X is part of the cursor-right escape sequence.

The other method of simulating "column only" positioning is less desirable, but may be effective in some instances. It uses the dummny cursor address variable Z in place of the Y address in a normal X-Y cursor address code (e.g., ESC "=" Z X).

CLEAR SCREEN & HOME @(-1)

The Clear Screen & Home code may consist of two different terminal control sequences (one for clear screen, and one for home). This is the case for VT-100 type terminals. Many other terminals combine these into one control sequence.

12.10 TEST-CURSOR

The TEST-CURSOR verb tests many of the cursor control definitions.

FORMAT:

```
TEST-CURSOR
```

TEST-CURSOR tests the following cursor control definitions:

@(x,y)	Position cursor at column x, row y
@(x)	Position cursor at column x in current row
@(-1)	Clear screen and home cursor
@(-2)	Home cursor
@(-3)	Clear from cursor positon to the end of the screen
@(-4)	Clear from cursor position to the end of the line
@(-5)	Blink on
@(-6)	Blink off
@(-7)	Start protected field
@(-8)	Stop protected field
@(-9)	Backspace cursor one character
@(-10)	Move cursor up one line
@(-11)	Enable protect mode
@(-12)	Disable protect mode
@(-13)	Reverse video on
@(-14)	Reverse video off
@(-15)	Underline on
@(-16)	Underline off
@(-19)	Move cursor right
@(-20)	Move cursor down

12.11 OPTIMIZING PARALLEL PRINTER PERFORMANCE

A verb, SET-LPTR, has been provided that allows several parameters to be specified that affect printer performance.

FORMAT:

SET-LPTR {p{,{delay},{burst},{slice}}

SET-LPTR is used to specify parameters for parallel printers in order to optimize system performance. The values for the parameters depend on the printer type, the number of users on the system, and the aspect of system performance that is most important. Some values increase the amount of work the printer can do, but lengthen the time it takes for the system to respond to users. Other values shorten the time it takes for the system to respond to users, but decrease the amount of work the printer can do. It is recommended that in order to find the optimum value for each system, several values be tested.

The parameters are as follows:

p — printer number; may be in range 0-3

delay — number of times to attempt to send a character to the printer; may be in range 1-4095. This number relates to the speed of the printer and the number of users that are sending jobs to the printer. If the number is too small, the printer performance degrades, but if it is too large, response time for users degrades.

The default is 300.

burst — number of characters sent to the printer during one transmission; may be in range 1-255. This number relates to the size of the printer buffer; in general, the value 15 provides good performance.

The default is 200.

slice — number of time slices the printer waits before receiving its next time slice; may be in range 1-15. This number relates to the number of users and number of printers on the system; in general, a number equal to or greater than the number of users provides good performance.

The default is 1.

If no parameters are specified, the current settings for all printers are displayed. If only the printer number is specified, the current settings for that printer are displayed.

If parameters are entered, only those that are being changed need to be specified; however, commas are needed to indicate missing values.

Statement	Description
>SET-LPTR 0 Printer 0 delay = 300 burst = 200 slice = 1	Displays printer settings for printer 0.
>SET-LPTR 1,,,5 Printer 1 delay = 300 burst = 200 slice = 5	Sets the slice for printer 1 to 5.

12.12 DOS TO PICK BRIDGE : COPYDOS

The COPYDOS verb allows data contained in the DOS partition to be transferred into the PICK partition.

FORMAT:

```
>COPYDOS dospath {(options{)}}
TO:(filename {itemname}
```

The syntax for the dospath parameter is as follows:

```
drive:\{subdirectory\...}dosfilename
```

For example:

```
drive:\subdirectory1\subdirectory2\dosfilename
          or
drive:\dosfilename
```

OPTIONS	MEANING
D	Display diagnostic information during translation.
F	Flag characters - system prompts for specifics.
L	Create indirect item (List Type).
M	Make multiple items - system prompts for specifics.
O	Overwrite existing PICK item.
P	Parity strip (characters converted to values 0-127). Flagged or Translated characters are not stripped.
R	DOS data is in random file mode.
S	DOS data is in sequential file mode (default).
T	Translate characters - system prompts for specifics.
X	Create hexadecimal image of DOS file.

If neither R nor S is specified in the options, the S option is assumed.

If neither the F nor T option is specified, the COPYDOS process will translate the DOS character X'0D' to X'FE'. This causes every DOS line delimited with a carriage return to become a PICK attribute. The linefeed X'0A' and null X'00' characters are deleted.

The D, F, M, T, and X options are explained further in the following section.

If the optional PICK itemname is not input, following the ' TO:(' prompt, the DOS filename (from the dospath parameter) is used as the PICK itemname.

Note also, that if the DOS file is larger than 25000 bytes and the M option has not been specified, the COPYDOS process will automatically

split the file into PICK items and assign the names itemname0, itemname1, etc. - unless the L option has been specified.

THE D OPTION

The D option displays diagnostic information while accessing the DOS file system. It is intended to be used as a diagnostic tool when data transfer operations do not complete as expected. If a lower case 'd' option is specified, the diagnostic information will be routed to the spooler.

THE F OPTION - FLAG CHARACTERS

The 'F' option is available to Flag up to 16 different hex bytes by placing a hex'00' in front of the Flagged hex character.

Entering in different hex bytes in response to the 'REPLACE:' and 'WITH FLAGGED' prompts, results in a Translation, with the resultant PICK byte preceded with hex'00'.

Entering in the same hex byte in response to both prompts, results in passing that byte unchanged and preceding it with hex'00'. Note that this is in contrast with the T option, where specifying the same character, deletes that character.

If an 'F' option is in effect, upon entering the ' TO:(' response the system will display:

REPLACE:

After entering the hex character to translate or pass, the system displays:

WITH FLAGGED:

Upon entry of the same character or a replacing character, the system will prompt for up to 15 additional hex characters to Flag. The system will prompt for additional characters, until a <CR> is entered at the REPLACE: prompt. A <CR> causes the system to display:

OKAY(Y/N):

An 'N' response allows re-entry of the FLAG specifications.

(Note: Should multiple occurrences of the same replace character be present, the last occurrence will take precedence.)

THE M OPTION - MULTIPLE PICK ITEMS

The M option allows for regulation of the size of the targeted PICK items. This flexibilty can be very useful in aligning DOS data with PICK attributes.

After entering a response to the ' TO:(' prompt the system displays:

ENTER LENGTH OF RECORD (OR Dn):

The user may elect to enter a numeric response indicating how many bytes each data portion of the PICK item will be, or a 'D' followed by a number to indicate how many PICK attributes (lines) each PICK item will be.

Using the 'Dn' response assumes that the default translate of carriage return/line feed to attribute marks will be performed.

When the DOS file is converted to multiple items, the items will be created with boundries established by attribute marks, provided that they exist in the translated file. Should no attribute marks exist, items will contain a single attribute with the specified number of bytes.

The PICK item-ids are generated by concatenating the itemname used in the ' TO:(' specification with 0 , 1 , 2 , etc. No new PICK item with just the itemname alone will be created.

Entering a <CR> after the appropriate response causes the COPYDOS process to begin or, if an F or T option is in effect, further prompting as noted below.

THE T OPTION - TRANSLATING CHARACTERS

The 'T' option is available to translate up to 16 different hex bytes. Upon entering a response to the ' TO:(' prompt, the system will display:

REPLACE:

After entering the hex character to replace, the system displays:

WITH:

Entering identical hex strings in response to both REPLACE: and WITH:, causes that character to be deleted from the file.

After entering a response to the WITH: prompt, the system will again display:

REPLACE:

Entering a <CR> after the REPLACE: prompt, terminates further hex character prompting, and the system displays:

OKAY(Y/N):

An 'N' response allows the user to re-enter all of the Translate specifications.
A 'Y' response causes the COPYDOS process to begin.

THE X OPTION - HEXADECIMAL IMAGE OF DOS FILE

The X option transfers data from the DOS file to the Pick environment and stores a hexadecimal image of the DOS file. This option allows data such as graphics images to be stored in a Pick item without concern for any special character (segment marks for instance, Hex 'FF').

COPYDOS Example

In the following example session, note that whenever multiple items are produced, (even without the M option), that an item with the actual PICK itemname used in the specification does not exist. The first item is the specified itemname with a zero appended.

```
>COPYDOS C:\SUB1\SUB2\DOSFILE (SMT

        TO:( PROCLIB PICK.SIDE

      ENTER LENGTH OF RECORD (OR Dn):  D5

          REPLACE:  0D                 WITH:  FE
          REPLACE:  0A                 WITH:  0A
          REPLACE:  2C                 WITH:  2A
          REPLACE:  <CR>

          OKAY(Y/N):  Y

          READING DIRECTORY
                 SUB1
                 SUB2
                 DOSFILE
          WRITING ITEM
                 PICK.SIDE0
                 PICK.SIDE1
                 PICK.SIDE2
                 PICK.SIDE3
         END OF FILE
```

Sample usage of the COPYDOS utility with the S, M, and T options.

12.13 PICK TO DOS BRIDGE : COPYPICK

The COPYPICK utility allows data contained in the PICK partition to be transferred into the DOS partition.

Since a PICK to DOS transfer is done from a DOS partition, a DOS diskette (available from your PC dealer) contains this utility. It is loaded with the command:

```
COPY A:*.* C:
```

This loads the PICK to DOS bridge program, COPYPICK.EXE into the DOS drive C:

FORMAT:

```
C>COPYPICK
```

After COPYPICK is invoked, the following prompt is displayed:

```
Options:
```

The available options include

- D Diagnostic mode operation has two levels. If the letter D is the first character of the option position string, a summary set of diagnostic messages is displayed. If the letter D is in option position 2 or greater, a more detailed set of diagnostics is presented.

- I Include Item-id as a line within the DOS file. The Item-id is always preceded with a line feed character. This option may be of particular use when moving all items in a file with the intent of doing additional processing in the DOS environment. The beginning of an item may be detected by the presence of a line feed either at the beginning of the file or immediately following another line feed.

- N Numeric Item-ids are assumed. This allows all items within a file to be moved to a DOS file in numeric order. Rather than a prompt for the Item-id, a range is requested. All items within this range are accessed out of the specified file. Should an item not be present, it is ignored and the process continues.

If desired, enter the appropriate option. If no options are desired, press <RETURN>.

The following prompts are displayed:

```
PICK Account Name:
PICK File (DICT FILE or FILE or FILE,DATA):
DOS File
PICK Item or * for all:
```

The DOS file name may be any valid DOS file on any valid DOS device. The DOS file prompt occurs before the request for the Item-ids so several items may be stored in the same DOS file. This is done by specifying individual Item-ids. A null Item-id indicates the end of input and precludes transferring an item with a null item-id. An asterisk (*) indicates all items within a file are to transferred. This means that a single item with an Item-id of asterisk cannot be transferred.

When transferring single items, the message displaying the number of items transferred and the number of records/attributes will accumulate and display the totals transferred to the DOS file.

If an invalid name is entered at any time during the prompt sequence, an appropriate error message, such as the following, is displayed:

```
File xxx in account yyy not found
```

A successful transfer ends with the following message:

```
n Item(s) Converted
n Attribute(s) Moved
```

After a successful transfer, the user will be prompted for another PICK item name. A <RETURN> displays the statistics for the last transfer, then returns to the prompt for the PICK file name. A <RETURN> at the file name prompt displays the account name prompt. A <RETURN> at the account prompt exits the COPYPICK utility.

PICK attribute marks (X'FE') are replaced with a carriage return (X'0D') and line-feed (X'0A'). This effectively makes each PICK attribute value a DOS record in the DOS receive file.

```
C>COPYPICK

Options : <RETURN>
PICK Account Name : SYSPROG
PICK File (DICT FILE or FILE or FILE,DATA) : BP
DOS File FINDX
PICK Item or * for all : FIND
      1 Item(s) Converted
     47 Attribute(s) Moved
PICK Item or * for all : <RETURN>
      1 Item(s) Converted
     47 Attribute(s) Moved
PICK File (DICT FILE or FILE or FILE,DATA) : <RETURN>
PICK Account Name : <RETURN>

C>
```

Sample COPYPICK Transfer Session.

'c' Table of Contents 'lc'

Chapter 13

PICK TUTORIAL

THE PICK PC SYSTEM

TUTORIAL

Contents

13 PICK TUTORIAL

13.1 INTRODUCTION 13-3
13.2 LOGON 13-4
13.3 TCL or TERMINAL CONTROL LANGUAGE 13-5
13.3.1 ACCESS 13-6
13.4 ACCESS SENTENCE SYNTAX 13-10
13.4.1 SORTING WITH ACCESS 13-11
13.4.2 DESCENDING SORTS WITH ACCESS 13-12
13.5 CONTROL BREAKS WITH ACCESS 13-13
13.5.1 HEADINGS & FOOTINGS WITH ACCESS 13-14
13.6 HEADING & FOOTING OPTIONS 13-14
13.6.1 TOTAL MODIFIER 13-16
13.7 SELECTION-CRITERIA: "WITH" 13-19
13.8 CREATING A FILE 13-20
13.9 DEFINING DICTIONARY ATTRIBUTES 13-21
13.9.1 BUILD.DICT PROGRAM 13-21
13.10 INPUT.DATA PROGRAM 13-23

13.1 INTRODUCTION

This tutorial will guide the new user of PICK through the basic operations of the system.

This tutorial will allow the new user to:

Learn how to log on to the system
Get data from the files produced into meaninful reports
Create data files
Define data file content

The PICK System uses some terms that are a little different than those used by most systems in data processing. Basic comparisons could be:

PICK SYSTEM	OTHER SYSTEMS
Item	Record
Item-id	Key
Attribute	Field

13.2 LOGON

When the system is booted up, or turned on, it verifies the operating system and has the user enter the time and date. Time is entered as 24-hour military time, therefore, when it is 2:00 PM, the time is entered as 14:00. The date is entered as MM-DD-YY.

The system will then display a message that it is verifying the system. This means that it is checking each frame of system code to make sure that everything is the way it should be.

When this is complete, the system will display:

Logon

Logging on is the means by which an account is entered. You must be logged on to an account in order to do any data processing. Every account has a Master Dictionary which contains all file names for that account and verbs that will cause the system to perform specific actions.

The name of the account that will be used to learn the system is called TUTOR.

To enter the PC-XT Tutorial Account, key in:

TUTOR <CR>

NOTE: <CR> means to press the Carriage Return key.

The screen will display:

PASSWORD:

The password for the TUTOR account is LEARN. When this is keyed in the system will not display it. If keyed in incorrectly, the system will return with the message:

USER ID?

Key in TUTOR again, and carefully key in LEARN for the password.

The system will enter the account, and display the TCL prompt (>).

13.3 TCL or TERMINAL CONTROL LANGUAGE

The TUTOR account is set up to display the TCL prompt. Unless the system is set up to automatically enter a program or another process the user will be at TCL.

TCL is shorthand for Terminal Control Lanuage. The user can tell if they are at TCL because a ">" prompt displays at the left-hand side of the screen. TCL is the basic way that the user communicates with the PICK Operating System.

Verbs can be entered at TCL to access files. Verbs are commands such as LIST, SORT, COPY, etc. TCL commands are not executed until a carriage return (noted as <CR>) is entered.

Although the tutotrial is primarily concerned with ACCESS commands, it should be pointed out that there are a variety of other commands which are also entered at the TCL prompt. All of them may be found in the PICK USER'S MANUAL. A few useful ones are noted here. The user may enter these commands any time the system is displaying the TCL prompt (>).

>TIME Outputs system time and date.

>LISTU List all ports (users) currently logged on the system.

>POVF Display the available space (frames) on the disk.

>LISTFILES List the files on this account.

The user should consult the PICK USER'S MANUAL for additional options and commands which are available at TCL.

13.3.1 ACCESS

ACCESS is a powerful, yet easy-to-use database retrieval language. ACCESS commands are entered at TCL. Let's enter a simple ACCESS command to see how the system works:

```
>LIST CUST
```

Where:

LIST	is the ACCESS verb to list a file
CUST	is the name of the file

The screen will display:

```
PAGE   1                                                    17:53:38
21 JAN 1985

CUST : 1006
COMPANY    TRACK AUTOMOTIVE
CONTACT    JACK NORTON
ADDRESS    7812 MAIN STREET
CITY    NEWARK
STATE    NJ
ZIP    07182
TELEPHONE    (206) 555-8347
INV#    17254    23846    48776    49003    55241
AMT    $11.27    $392.72    $371.82    $984.84    $93.89
DATE1    15  DEC  1984    31 DEC  1984    15 JAN 1985    25 JAN  1985
01 APR 1985

CUST : 1000
COMPANY    ACME HARDWARE COMPANY
CONTACT    JOHN THOMPSON
ADDRESS    1134 BRISTOL PKWAY
CITY    IRVINE
STATE    CA
ZIP    92714
TELEPHONE    (714) 555-9384
```

The system will display 22 lines of data on the screen. Of course, most reports are more than a screenful, so to see the remainder, key a <CR>. If one screenful is enough information, then press the <CNTL> and <X> keys simultaneously. The listing will terminate and the system will return to TCL.

The definitions on the attributes that were displayed are:

```
CUST#      - The item-id of this item in the CUST file
COMPANY    - The first attribute, defined as both "1" and
"COMPANY"
CONTACT    - The second attribute, defined as both "2" and
"CONTACT'
ADDRESS    - The third attribute, defined as both "3" and
"ADDRESS"
CITY       - The fourth attribute, defined as both "4" and CITY
STATE      - The fifth attribute, defined as both "5" and STATE
ZIP        - The sixth attribute, defined as both "6" and ZIP
TELEPHONE  - The seventh attribute, defined as both "7" and
TELEPHONE
INV#       - The eighth attribute, defined as both "8" and INV#
AMT        - The ninth attribute, defined as both "9" and AMT
DATE1      - The tenth attribute, defined as both "10" and DATE1
```

The attributes are defined in a section of the file called the File "DICT". Each file has a dictionary section and a data section. When a LIST command is given the system will look at the file dictionary and display all sequential fields defined as 1, 2, 3, etc. If these have not been defined, then the system will display only the item-id for each item in the file.

The attributes INV#, AMT and DATE1 are called multi-valued attributes. In other words, there is more than one value in the field. Attributes can be broken down into values and these values further broken down into sub-values to define data.

This CUST file has synonym attributes defined. This means that whether the field is entered as "1" or "COMPANY" the result will be the same, because both are defined exactly the same way in the dictionary of CUST.

The user may define any number of synonym definitions, which are helpful because the user doesn't have to remember which attribute is number four, but can call up the data by specifying a meaninful synonym (e.g., "CITY" or "SHIP.TO.CITY", etc.).

If a listing was to be sorted by the Customer Number, the command would be:

```
>SORT CUST <CR>
```

The screen will display:

```
PAGE   1                                                    17:53:11
21 JAN 1985

CUST : 1000
COMPANY    ACME HARDWARE COMPANY
CONTACT    JOHN THOMPSON
ADDRESS    1134 BRISTOL PKWAY
CITY    IRVINE
STATE    CA
ZIP    92714
TELEPHONE    (714) 555-9384
INV#    48372    49182    50192    51327    82712
AMT    $512.13    $439.98    $283.47    $283.74    $182.73
DATE1    28  OCT 1984     18  NOV 1984    15 DEC 1984     31 DEC 1984
15 JAN 1985
```

```
CUST : 1001
COMPANY    NEWTON DEVELOPMENT
CONTACT    THOMAS NEWTON
ADDRESS    1970 SKYLARK STREET
CITY    HUNTINGTON BEACH
STATE    CA
ZIP    92785
TELEPHONE    (714) 555-9283
```

Notice that this time, the difference is that the CUST items are sorted in order.

It should be evident that these two commands are very similar. The real difference being that one lists the items in the order they are stored on the disk, and the other sorts the items by the item-id.

13.4 ACCESS SENTENCE SYNTAX

ACCESS is an English-like inquiry language. Each ACCESS "sentence" must consist of a verb followed by a file name.

A verb is an action-oriented word which will invoke a specific ACCESS processor. The LIST CUST statement above is an example of the simplest form of an ACCESS command. (See 6.2 ACCESS verbs).

However, ACCESS commands may be made even more useful, using the verb and file name and then adding selection criteria, sort keys, output specifications and print limiters, to get custom reports. (See 6.4 Rules for generating ACCESS sentences).

To list the file and only see certain attributes, key in:

```
>LIST CUST COMPANY CITY STATE <CR>
```

The screen will display:

```
PAGE   1                                                    17:54:06
21 JAN 1985

CUST..... COMPANY.................. CITY.............. STATE

1006      TRACK AUTOMOTIVE         NEWARK              NJ
1000      ACME HARDWARE COMPANY    IRVINE              CA
1007      MESA TRAVEL AGENCY       HUNTINGTON BEACH    CA
1001      NEWTON DEVELOPMENT       HUNTINGTON BEACH    CA
1008      WORD ALBEGRA             CHICAGO             IL
1002      UPTOWN PRINTERS          LOS ANGELES         CA
1009      MY TIMES MAGAZINE        NEWARK              NJ
1003      RITE-WAY DRUGS           CHICAGO             IL
1010      PICK SYSTEMS             IRVINE              CA
1004      LIKE-NU UPHOLSTERY       CHICAGO             IL
1005      A-1 APPLIANCES           NEWARK              NJ

11 ITEMS LISTED.
```

NOTE: >LIST CUST 1 4 5 <CR> would have produced an identical listing.

(See 6.26 LIST verb)

Notice that this time the data displayed across the screen in a horizontal fashion rather than down the page as in the first two listings. This is because a screen can only display 79 characters. ACCESS will check to see if the generated report will be wider than 79 characters. If it is, then the listing is done vertically. If the listing fits into 79 characters, the system will list the data horizontally as shown above.

13.4.1 SORTING WITH ACCESS

To produce a report that lists only certain fields, is sorted by zip code and lists only the company name, city and zip code, enter:

```
>SORT CUST BY ZIP COMPANY CITY ZIP
```

Where:

SORT	is the ACCESS verb
CUST	is the file name
BY ZIP	is the attribute to sort by
COMPANY	is the first attribute to display.
CITY	is the second attribute to display.
ZIP	is the third attribute to display.

The screen will display:

```
PAGE   1                                                    18:02:00
21 JAN 1985

CUST..... COMPANY.................. CITY.............. ZIP..

1005      A-1 APPLIANCES           NEWARK              07152
1006      TRACK AUTOMOTIVE         NEWARK              07182
1009      MY TIMES MAGAZINE        NEWARK              07273
1008      WORD ALBEGRA             CHICAGO             60611
1003      RITE-WAY DRUGS           CHICAGO             60623
1004      LIKE-NU UPHOLSTERY       CHICAGO             60681
1002      UPTOWN PRINTERS          LOS ANGELES         90099
1007      MESA TRAVEL AGENCY       HUNTINGTON BEACH    92647
1000      ACME HARDWARE COMPANY    IRVINE              92714
1010      PICK SYSTEMS             IRVINE              92714
1001      NEWTON DEVELOPMENT       HUNTINGTON BEACH    92785

11 ITEMS LISTED.
```

NOTE: >SORT CUST BY 6 1 4 6 <CR> would have produced an identical listing.

(See 6.27 SORT verb)

13.4.2 DESCENDING SORTS WITH ACCESS

If the report was to be sorted with zip codes in the order 99999 to 00001, instead of the usual 00001 to 99999, the ACCESS command would be:

```
>SORT CUST BY-DSND ZIP COMPANY CITY ZIP

PAGE   1                                                    17:55:05
21 JAN 1985

CUST..... COMPANY.................. CITY.............. ZIP..

1001      NEWTON DEVELOPMENT       HUNTINGTON BEACH    92785
1000      ACME HARDWARE COMPANY    IRVINE              92714
1010      PICK SYSTEMS             IRVINE              92714
1007      MESA TRAVEL AGENCY       HUNTINGTON BEACH    92647
1002      UPTOWN PRINTERS          LOS ANGELES         90099
1004      LIKE-NU UPHOLSTERY       CHICAGO             60681
1003      RITE-WAY DRUGS           CHICAGO             60623
1008      WORD ALBEGRA             CHICAGO             60611
1009      MY TIMES MAGAZINE        NEWARK              07273
1006      TRACK AUTOMOTIVE         NEWARK              07182
1005      A-1 APPLIANCES           NEWARK              07152

11 ITEMS LISTED.
```

NOTE: >SORT CUST BY-DSND 6 1 4 6 <CR> would have produced an identical listing.

BY-DSCD tells ACCESS to sort this attribute in descending order, 9-0 for numbers and Z-A for alphabetic characters.

(See 6.27.1 BY-DSND modifier)

13.5 CONTROL BREAKS WITH ACCESS

If the list is to be separated into categories, for instance city, then the command would be:

```
>SORT CUST BY CITY COMPANY BREAK-ON CITY

PAGE   1                                                    19:43:33
21 JAN 1985

CUST..... COMPANY..................... CITY.................

1003       RITE-WAY DRUGS             CHICAGO
1004       LIKE-NU UPHOLSTERY         CHICAGO
1008       WORD ALBEGRA               CHICAGO

                                      ***

1001       NEWTON DEVELOPMENT         HUNTINGTON BEACH
1007       MESA TRAVEL AGENCY         HUNTINGTON BEACH

                                      ***

1000       ACME HARDWARE COMPANY      IRVINE
1010       PICK SYSTEMS               IRVINE

                                      ***

1002       UPTOWN PRINTERS            LOS ANGELES

                                      ***

1005       A-1 APPLIANCES             NEWARK
1006       TRACK AUTOMOTIVE           NEWARK
1009       MY TIMES MAGAZINE          NEWARK

                                      ***

***

11 ITEMS LISTED.
```

NOTE: >SORT CUST BY 4 1 BREAK-ON 4 <CR> would have produced an identical listing.

(See 6.25 CONTROL BREAKS)

13.5.1 HEADINGS & FOOTINGS WITH ACCESS

Reports may have either a heading or a footing so that the person reading it can readily ascertain what report they are looking at.

To this end, ACCESS has HEADING and FOOTING directives. Key in the following ACCESS sentence:

```
>SORT CUST COMPANY CITY STATE HEADING "MY FIRST PICK
 REPORT 'CL' TODAYS DATE IS 'DCL' PAGE 'PCL'"
```

The result should look like:

```
                        MY FIRST PICK REPORT
                     TODAYS DATE IS 23 JAN 1985
                             PAGE    1

CUST..... COMPANY..................  CITY.............  STATE

1000      ACME HARDWARE COMPANY      IRVINE             CA
1001      NEWTON DEVELOPMENT         HUNTINGTON BEACH   CA
1002      UPTOWN PRINTERS            LOS ANGELES        CA
1003      RITE-WAY DRUGS             CHICAGO            IL
1004      LIKE-NU UPHOLSTERY         CHICAGO            IL
1005      A-1 APPLIANCES             NEWARK             NJ
1006      TRACK AUTOMOTIVE           NEWARK             NJ
1007      MESA TRAVEL AGENCY         HUNTINGTON BEACH   CA
1008      WORD ALBEGRA               CHICAGO            IL
1009      MY TIMES MAGAZINE          NEWARK             NJ
1010      PICK SYSTEMS               IRVINE             CA

11 ITEMS LISTED.
```

NOTE: >SORT CUST 1 4 5 HEADING... <CR> would have produced an identical listing.

(See 6.20 HEADINGS & FOOTINGS)

13.6 HEADING & FOOTING OPTIONS

The date on the report will be the current date that the report is run. The heading text must be enclosed in double quotes (") after the word heading. The mnemonics enclosed in single quotes (') must be within the HEADING double quotes and is telling the system:

- C - Center the line
- L - perform a Line feed
- D - todays Date
- P - incrementing Page number

There are other parameters that can be used in a heading or footing. The only difference between a HEADING and a FOOTING

directive is that a heading prints at the top of each page and a footing at the bottom. Otherwise, the way they are used is exactly the same. ACCESS can generate both a heading and a footing on the same page.

(See 6.20 HEADINGS & FOOTINGS)

13.6.1 TOTAL MODIFIER

To total specific attribute values, consider the following example:

```
>SORT CUST BREAK-ON COMPANY TOTAL AMT INV# DATE1
<CR>
```

WHERE:

SORT	is the ACCESS verb
CUST	is the file name
BREAK-ON	causes a break-on output when value changes
COMPANY	is the attribute to BREAK-ON
TOTAL	totals all values of following attribute upon break
AMT	is the attribute containing the values to be totaled
INV#	is an attribute to display
DATE1	is an attribute to display

The result should be:

```
PAGE   1                                                    20:04:18
21 JAN 1985

CUST..... COMPANY.................. AMT... INV#..... DATE1...
1000      ACME HARDWARE COMPANY    $512.13 48372   28 OCT 1984
                                   $439.98 49182   18 NOV 1984
                                   $283.47 50192   15 DEC 1984
                                   $283.74 51327   31 DEC 1984
                                   $182.73 82712   15 JAN 1985

          ***                    $1,702.05

1001      NEWTON DEVELOPMENT       $489.38 18473   28 SEP 1984
                                   $384.98 28374   27 OCT 1984
                                   $184.89 39475   15 NOV 1984
                                   $852.43 48567   20 DEC 1984
                                   $348.78 50572   25 JAN 1985

          ***                    $2,260.46

1002      UPTOWN PRINTERS          $453.34 19573   23 NOV 1984
                                   $118.40 20773   24 JAN 1985
                                   $394.68 36574   20 MAR 1985
                                   $938.73 40112   01 APR 1985
                                   $384.06 50283   23 APR 1985

          ***                    $2,289.21

1003      RITE-WAY DRUGS           $105.48 19567   28 OCT 1984
                                   $483.05 29845   15 DEC 1984
                                   $306.74 38945   31 DEC 1984
                                   $384.62 46355   25 JAN 1985
                                   $483.84 59375   02 APR 1985

          ***                    $1,763.73
```

```
1004        LIKE-NU UPHOLSTERY        $437.34 19573   18 NOV 1984
                                      $823.78 28563   15 DEC 1984
                                      $192.82 39573   31 DEC 1984
                                      $981.82 48956   24 JAN 1985
                                      $815.31 51113   01 APR 1985

            ***                     $3,251.07

1005        A-1 APPLIANCES            $182.71 18372   18 NOV 1984
                                       $45.81 18299   15 DEC 1984
                                      $293.73 20478   31 DEC 1984
                                      $102.87 40394   25 JAN 1985
                                       $18.72 49907   26 FEB 1985

            ***                       $643.84

1006        TRACK AUTOMOTIVE           $11.27 17254   15 DEC 1984
                                      $392.72 23846   31 DEC 1984
                                      $371.82 48776   15 JAN 1985
                                      $984.84 49003   25 JAN 1985
                                       $93.89 55241   01 APR 1985

            ***                     $1,854.54

1007        MESA TRAVEL AGENCY      $1,927.00 10287   28 OCT 1984
PAGE   2                                                    20:04:26
21 JAN 1985

CUST..... COMPANY.................. AMT... INV#..... DATE1...
                                       $29.19 20389   23 FEB 1985
                                      $293.98 30816   29 MAR 1985
                                      $239.29 41776   31 DEC 1984
                                      $192.93 52891   09 MAR 1985

            ***                     $2,682.39

1008        WORD ALBEGRA              $382.81 19673   24 SEP 1984
                                      $912.23 29837   29 JAN 1985
                                      $391.93 39478   10 MAR 1985
                                      $938.71 55982   09 APR 1985
                                      $938.03 59102   26 JUN 1985
                                                      10 JUL 1985

            ***                     $3,563.71

1009        MY TIMES MAGAZINE          $12.82 10284   13 SEP 1984
                                      $239.39 20678   22 OCT 1984
                                      $391.90 34817   19 DEC 1984
                                      $563.32 59836   13 FEB 1985
                                      $168.23 59902   26 JUN 1985
                                                      08 JUL 1985

            ***                     $1,375.66

1010        PICK SYSTEMS              $38.18 19573   14 SEP 1984
                                      $28.19 22014   28 OCT 1984
```

```
                                  $349.53 34001   15 DEC 1984
                                  $493.61 48900   31 DEC 1984
                                   $10.53 52261   27 OCT 1984

          ***                     $920.04

***                             $22,306.70

11 ITEMS LISTED.
```

(See 6.21 TOTAL MODIFIER)

13.7 SELECTION-CRITERIA: "WITH"

To make a selection from one of the attributes of the file, the ACCESS command line could be:

```
      >SORT CUST BY CITY COMPANY CITY ZIP WITH CITY  = "CHICAGO"
<CR>
```

The result should be:

```
PAGE   1                                                    09:35:28
24 JAN 1985

CUST..... COMPANY.................. CITY.............

1003       RITE-WAY DRUGS          CHICAGO
1004       LIKE-NU UPHOLSTERY      CHICAGO
1008       WORD ALBEGRA            CHICAGO

3 ITEMS LISTED.
```

(See 6.10 SELECTION-CRITERIA)

Experiment with the CUST file and ACCESS commands. More detailed explanations and other commands may be found in the ACCESS chapter of the PICK USER REFERENCE MANUAL.

13.8 CREATING A FILE

All data on the PICK System is in files. Data files have two portions to them, the DICT portion and the DATA portion.

The DICT of a file has all of the attributes of the data file defined in it. The DICT controls how many characters to allocate an attribute upon output, whether it is left- or right-justified on a report, the column heading to print for an attribute on a report and other paramemters.

The DATA portion of a file contains the data. All of the ACCESS commands in the examples have been run against the data portion of the file CUST.

The PICK System stores data on the disk in "frames". A frame is 512 bytes (or characters). If one frame is full, then the system will automatically attach another frame to it so that a file can "grow" naturally.

When a file is created, the user must specify how many frames should be initially allocated for the DICT and DATA portions of the file. This is generally figured by how many characters there are going to be in an item (record) and how many items will be in the file.

Let's create a file called NAMES and then define what the fields will be and enter data. To do this, key in:

```
>CREATE-FILE NAMES 3 7 <CR>
```

WHERE:

CREATE-FILE	is the TCL command to create a file
NAMES	is the file name
3	is the number of frames to allocate to the DICT
7	is the number of frames to allocate to the DATA portion of the file

The numbers 3 and 7 indicate the number of frames to be reserved for the ' DICT and DATA portions of the file respectively. This is referred to as the MODULO of the file.

After the CREATE-FILE command is keyed in, the system will respond with:

```
[417] FILE 'NAMES' CREATED; BASE = 3017, MODULO = 3, SEPAR = 1.
[417] FILE 'NAMES' CREATED; BASE = 5200, MODULO = 7, SEPAR = 1.
```

The two lines that are returned by the system refer to the DICT and DATA portions respectively. BASE is the starting frame address, MODULO is how many frames were specified and SEPAR (separation) is always 1.

13.9 DEFINING DICTIONARY ATTRIBUTES

The DICT portion of the file has items that define what the data will be in the DATA portion of the file.

A PICK/BASIC program is on the TUTOR account that will allow you to define the DICT section of your NAMES file.

13.9.1 BUILD.DICT PROGRAM

To enter the Dictionary Definition program, key in:

```
>BUILD.DICT <CR>
```

The screen will prompt to enter the file name:

```
ENTER FILE NAME:  NAMES <CR>
```

If the file NAMES has not been created, then the system will return an error message that the file cannot be found. Return to the section on creating a file and create the NAMES file.

The entry screen will display:

```
A T T R I B U T E   D E F I N I T I O N   E N T R Y

File DICT is:  NAMES

This ITEM-ID is:  1

Enter attribute name/description:

Enter attribute justification (L, R, T, U):

Enter attribute output length:
```

The BUILD.DICT program will allow the definition of up to ten (10) attribute definitions. The Item-id for these definition items start with "1" and increments for each new definition entered (up to 10). In a name and address file the definitions would probably be:

DESCRIPTION	JUSTIFICATION	LENGTH
NAME	L	20
ADDRESS	L	20
CITY	L	20
STATE	L	2
ZIP	R	5

The BUILD.DICT program will prompt for the description, justification and output length for each attribute defining item.

Whatever is defined as the DESCRIPTION will be the column heading on a report that is produced through ACCESS.

JUSTIFICATION refers to whether the data should line up at the left or right of the field. Alpha/numeric data is generally specified as being left justified and numeric data is generally specified as being right justified. The difference for numeric fields is:

LEFT JUSTIFY	RIGHT JUSTIFY
1001	1001
101	101
10001	10001

The LENGTH refers to the column width of an attribute upon output. If data is entered in an attribute that is longer than the defined length there is no error. However, that data will "wrap" on a horizontally listed report if it has more characters in the attribute than was defined in the length. This should not be a problem if fields are realistically defined for the length of the data.

While using the dictionary attribute definition program, if there is a question on an input field, press "?" as the first character and a help screen will display for that attribute.

After the description, justification and length have been defined for an attribute definition item, the system will prompt:

ALL FIELDS CORRECT? (Y/N)

If the data entered suits you, then key an upper case "Y" and press the carriage return. If there needs to be a change, press "N" and the cursor will return to the description field to allow different entry.

If all data is correct and "Y" is entered, the system will prompt;

ENTER ANOTHER ATTRIBUTE DEFINITION (Y/N):

Enter a "Y" and press carriage return until all the fields desired have been defined.

The program will automatically increment the Item-ID by 1. Again, only ten (10) attributes may be defined using this program. When all attributes have been defined answer the last prompt with an "N" and the system will return to TCL.

13.10 INPUT.DATA PROGRAM

There is a program on the TUTOR account called INPUT.DATA. This program will allow you to enter information into the DATA portion of the NAMES file.

To use this program, key in:

>INPUT.DATA <CR>

The screen will prompt:

ENTER FILE NAME: NAMES <CR>

Key in NAMES and press carriage return. If the NAMES file has not been created the system will return an error message. Return to the section on creating a file and create the NAMES file.

The screen will display:

```
D A T A   E N T R Y   S C R E E N

File name is:  NAMES

Enter unique ID:
```

Below this will be the descriptions that were defined for the DICT NAMES using the BUILD.DICT program. If there are no prompts for inputting data fields, then return to the BUILD.DICT program to create them.

Each item (or record) MUST HAVE A UNIQUE ID. This can be almost anything the user wants. For the purposes here, we suggest you use 101, 102, 103, etc.

After the Item-ID is entered, press carriage return to go from line to line and enter the appropriate data. When finished, the system will display:

DO YOU WANT TO ENTER ANOTHER ITEM? (Y/N)

Press "Y" to enter another item into the NAMES file. Press "N" to signal that data entry is complete and the system will return to TCL.

Once the data has been entered, return to the section on ACCESS and try some of the commands on your data. Since the file dictionary contains sequential item-ids (1,2...10), the command "LIST NAMES" will default to show as many attributes as you have consecutively defined.

Appendices

THE PICK SYSTEM

USER MANUAL

Appendix A

DEFINING KEYBOARDS

Several keyboard definition items are included with the system. The character that is transmitted when a key is pressed depends on the following factors:

- the scan code that is generated
- the keyboard that is in effect
- the condition of the key
- whether a special 'lead-in' key was pressed

SCAN CODES

Each key generates a scan code when it is pressed. Figure A-1 shows the scan codes generated by an 84-key keyboard; Figure A-2 shows the scan codes generated by a 101-key keyboard; Figure A-3 shows the scan codes generated by a 102-key keyboard. These are representative keyboards; to determine the actual scan codes generated by a system, refer to the technical documentation for that system.

KEYBOARDS

Keyboards are defined for 84-, 101-, and 102-key keyboards as items in the file KEYBOARDS in the account SYSPROG and include:

BRITISH-84
BRITISH-101
FRENCH-84
FRENCH-102
GERMAN-84
GERMAN-102
ITALIAN-84
ITALIAN-102
SPANISH-84
SPANISH-102
USA-84
USA-101

The default keyboard is based on the USA 101-key keyboard as it is shipped. The actual default keyboard is coded into the PICK operating system and cannot be modified. The default keyboard is selected each time the system is booted. To change the keyboard to a different definition, use the SET-KBRD command. The keyboard definitions themselves can be modified by changing the item in the KEYBOARD file. Items can also be added or deleted from the KEYBOARD file.

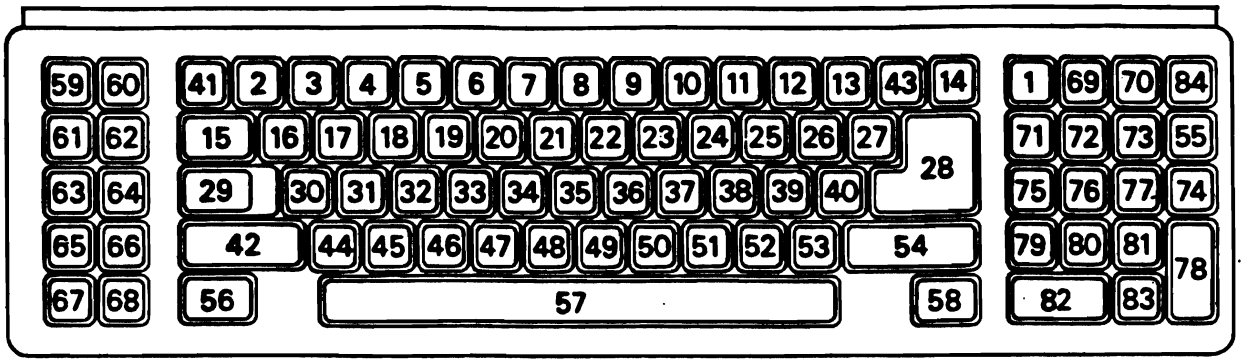


Figure 1 84-Keyboard Layout

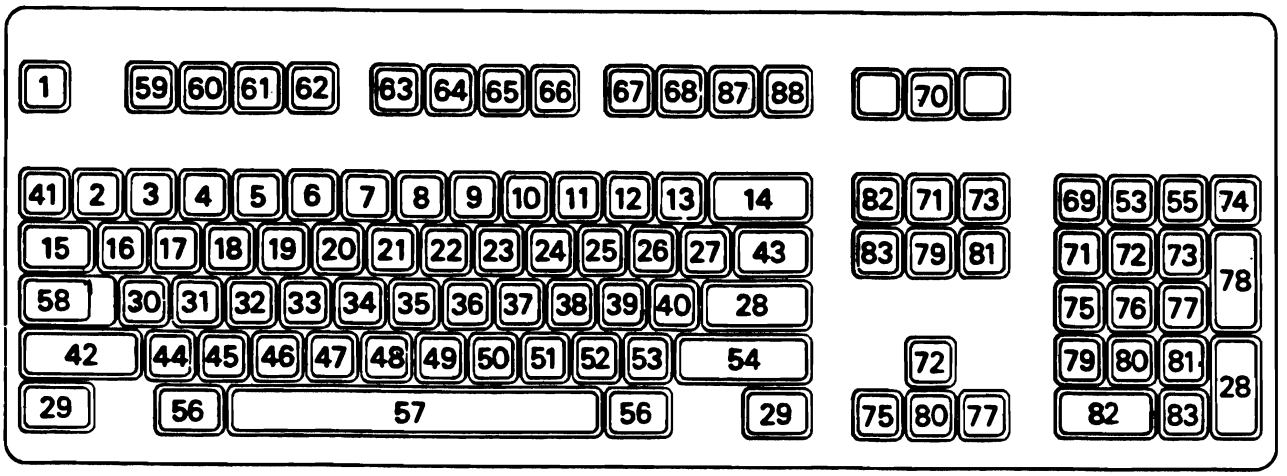


Figure 2 101-Keyboard Layout

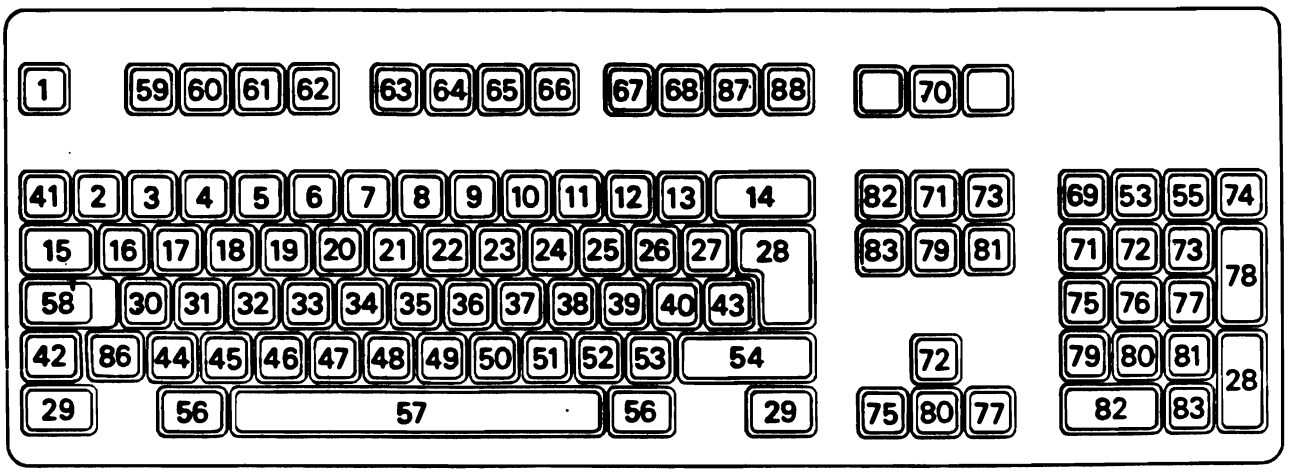


Figure 3 102-Keyboard Layout

CONDITION OF KEY

The condition of the key depends on whether the key is pressed by itself, with the <SHIFT> key, with the <CTRL> key, or with the <ALT> key. Each KEYBOARD item is made up of four sets of attributes that correspond to the four conditions of the key recognized by PICK.

The keyboard definitions for keys pressed by themselves are intended for lower case letters and characters displayed on the lower part of the key. The keyboard definitions for keys pressed with the <SHIFT> key are intended for upper case letters and characters displayed on the upper part of the key. The keyboard definitions for keys pressed with the <CTRL> key are intended for the standard ASCII control characters. The keyboard definitions for keys pressed with the <ALT> key are intended for the characters displayed on the vertical face of the keys.

In addition, the upper case definitions include the keys on the numeric keypad that are pressed when NUMLOCK is on; the lower case definitions include the keys on the numeric keypad that are pressed when NUMLOCK is off.

LEAD-IN CHARACTERS

Several alphabets use diacritical marks with certain vowels. These diacritical marks are supported by use of special lead-in character keys. When a lead-in character key is pressed, the system waits for the next key to be pressed before transmitting any character. If the next key that is pressed has a form with the diacritical mark, that character is displayed. If there is no form of the letter with a diacritical mark, nothing is displayed.

The following lead-in characters are supported:

Keyboard Character	Characters in KEYBOARD item	Character Name
o	FA	Degree
^	FB	Circumflex
'	FC	Accent Acute
`	FD	Accent Grave
	FE	Diaresis (umlaut)

NOTE: The values for the diacritical marks resemble PICK system delimiters; they are NOT delimiters, however. The values were chosen because no key normally generates these values.

GETTING THE CHARACTER

When a key is pressed, the system determines the scan code and the condition of the key. It then uses the scan code as an index into the item for the active keyboard. The keyboard is made up of four sets of attributes, one for each condition of the key.

Each set of attributes contains 128 entries, divided into eight attributes. Each attribute contains sixteen 2-digit hexadecimal numbers that define the characters to be generated. The first hex number corresponds to scan code 0, the next number to scan code 1, etc.

For example, assume that the key that generates scan code 16 is pressed by itself. The system looks up the sixteenth entry in the table for the lower case entries, that is, the first character in the second attribute in the set of attributes for lower case entries. If the USA keyboard is active, this produces the character equivalent to the hexadecimal value 71, which a lower case q. If the French keyboard is active, this produces the character equivalent to the hexadecimal value 61, which is a lower case a.

The tables used to create the items are listed in the following pages.

These definitions affect line 0 (memory-mapped monitor) only.

BRITISH 84-Key Keyboard

* Note: X'FF' means ignore this keystroke

*

Scan Codes	Upper Case Definitions
0-15	|FF|1B|21|22|9C|24|25|5E|26|2A|28|29|5F|2B|08|FF|
16-31	|51|57|45|52|54|59|55|49|4F|50|7B|7D|0D|FF|41|53|
32-47	|44|46|47|48|4A|4B|4C|3A|40|7C|FF|7E|5A|58|43|56|
48-63	|42|4E|4D|3C|3E|3F|FF|FF|FF|20|FF|50|51|52|53|54|
64-79	|55|56|57|58|59|FF|FF|37|38|39|2D|34|35|36|2B|31|
80-95	|32|33|30|2E|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
96-111	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
112-127	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

*

Lower Case Definitions

0-15	|FF|1B|31|32|33|34|35|36|37|38|39|30|2D|3D|08|09|
16-31	|71|77|65|72|74|79|75|69|6F|70|5B|5D|0D|FF|61|73|
32-47	|64|66|67|68|6A|6B|6C|3B|27|5C|FF|23|7A|78|63|76|
48-63	|62|6E|6D|2C|2E|2F|FF|2A|FF|20|FF|40|41|42|43|44|
64-79	|45|46|47|48|49|FF|FF|FF|1A|FF|2D|15|FF|06|2B|FF|
80-95	|0A|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
96-111	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
112-127	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

*

Control Key Definitions

0-15	|FF|1B|FF|FF|FF|FF|FF|1E|FF|FF|FF|FF|1F|FF|7F|FF|
16-31	|11|17|05|12|14|19|15|09|0F|10|1B|1D|0A|FF|01|13|
32-47	|04|06|07|08|0A|0B|0C|FF|00|1C|FF|FF|1A|18|03|16|
48-63	|02|0E|0D|FF|FF|FF|FF|FF|FF|20|FF|20|21|22|23|24|
64-79	|25|26|27|28|29|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
80-95	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
96-111	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
112-127	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

*

Alternate Key Definitions

0-15	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
16-31	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
32-47	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
48-63	|FF|FF|FF|FF|FF|FF|FF|FF|FF|20|FF|30|31|32|33|34|
64-79	|35|36|37|38|39|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
80-95	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
96-111	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
112-127	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

```
*  Define BRITISH 101-key keyboard
*
*  Note: X'FF' means ignore this keystroke
*

*
```

Scan Codes	Upper Case Definitions
0-15	|FF|1B|21|40|23|24|25|5E|26|2A|28|29|5F|2B|08|FF|
16-31	|51|57|45|52|54|59|55|49|4F|50|7B|7D|0D|FF|41|53|
32-47	|44|46|47|48|4A|4B|4C|3A|22|7E|FF|7C|5A|58|43|56|
48-63	|42|4E|4D|3C|3E|3F|FF|2A|FF|20|FF|50|51|52|53|54|
64-79	|55|56|57|58|59|FF|FF|37|38|39|2D|34|35|36|2B|31|
80-95	|32|33|30|2E|FF|FF|FF|5A|5B|FF|FF|FF|FF|FF|FF|FF|
96-111	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
112-127	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

```
*
```

Lower Case Definitions	
0-15	|FF|1B|31|32|33|34|35|36|37|38|39|30|2D|3D|08|09|
16-31	|71|77|65|72|74|79|75|69|6F|70|5B|5D|0D|FF|61|73|
32-47	|64|66|67|68|6A|6B|6C|3B|27|60|FF|5C|7A|78|63|76|
48-63	|62|6E|6D|2C|2E|2F|FF|2A|FF|20|FF|40|41|42|43|44|
64-79	|45|46|47|48|49|FF|FF|FF|1A|FF|2D|15|FF|06|2B|FF|
80-95	|0A|FF|FF|FF|FF|FF|FF|4A|4B|FF|FF|FF|FF|FF|FF|FF|
96-111	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
112-127	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

```
*
```

Control Key Definitions	
0-15	|FF|1B|FF|00|FF|FF|FF|1E|FF|FF|FF|FF|1F|FF|7F|FF|
16-31	|11|17|05|12|14|19|15|09|0F|10|1B|1D|0A|FF|01|13|
32-47	|04|06|07|08|0A|0B|0C|FF|FF|FF|FF|1C|1A|18|03|16|
48-63	|02|0E|0D|FF|FF|FF|FF|FF|FF|20|FF|20|21|22|23|24|
64-79	|25|26|27|28|29|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
80-95	|FF|FF|FF|FF|FF|FF|FF|2A|2B|FF|FF|FF|FF|FF|FF|FF|
96-111	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
112-127	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

```
*
```

Alternate Key Definitions	
0-15	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
16-31	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
32-47	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
48-63	|FF|FF|FF|FF|FF|FF|FF|FF|FF|20|FF|30|31|32|33|34|
64-79	|35|36|37|38|39|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
80-95	|FF|FF|FF|FF|FF|FF|FF|3A|3B|FF|FF|FF|FF|FF|FF|FF|
96-111	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
112-127	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

```
*  Define FRENCH 84-key keyboard
*
*  Note: X'FF' means ignore this keystroke

*
```

Scan Codes — Upper Case Definitions

Scan Codes															
0-15	FF	1B	31	32	33	34	35	36	37	38	39	30	FA	5F	08	FF
16-31	41	5A	45	52	54	59	55	49	4F	50	FE	2A	0D	FF	51	53
32-47	44	46	47	48	4A	4B	4C	4D	25	3E	FF	9C	57	58	43	56
48-63	42	4E	3F	2E	2F	2B	FF	FF	FF	20	FF	50	51	52	53	54
64-79	55	56	57	58	59	FF	FF	37	38	39	2D	34	35	36	2B	31
80-95	32	33	30	2E	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

```
*
```

Lower Case Definitions

0-15	FF	1B	26	82	22	27	28	15	8A	21	87	85	29	2D	08	09
16-31	61	7A	65	72	74	79	75	69	6F	70	FB	24	0D	FF	71	73
32-47	64	66	67	68	6A	6B	6C	6D	FC	3C	FF	E6	77	78	63	76
48-63	62	6E	2C	3B	3A	3D	FF	2A	FF	20	FF	40	41	42	43	44
64-79	45	46	47	48	49	FF	FF	FF	1A	FF	2D	15	FF	06	2B	FF
80-95	0A	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

```
*
```

Control Key Definitions

0-15	FF	1B	00	FF	FF	FF	FF	1E	FF	FF	FF	FF	FF	1F	7F	FF
16-31	01	1A	05	12	14	19	15	09	0F	10	1B	1D	0A	FF	11	13
32-47	04	06	07	08	0A	0B	0C	0D	FF	1C	FF	FF	17	18	03	16
48-63	02	0E	FF	FF	FF	FF	FF	FF	FF	20	FF	20	21	22	23	24
64-79	25	26	27	28	29	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
80-95	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

```
*
```

Alternate Key Definitions

0-15	FF	FF	40	FF	23	FF	FF	5E	FF	FF	FF	FF	FF	FF	FF	FF
16-31	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	5B	5D	FF	FF	FF	FF
32-47	FF	FF	FF	FF	FF	FF	FF	FF	FF	5C	FF	FF	FF	FF	FF	FF
48-63	FF	FF	FF	FF	FF	FF	FF	FF	FF	20	FF	30	31	32	33	34
64-79	35	36	37	38	39	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
80-95	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

```
*  Define FRENCH 102-key keyboard
*
*  Note: X'FF' means ignore this keystroke

*
```

Scan Codes — Upper Case Definitions

Scan Codes															
0-15	FF	1B	31	32	33	34	35	36	37	38	39	30	FA	2B	08	FF
16-31	41	5A	45	52	54	59	55	49	4F	50	FE	9C	0D	FF	51	53
32-47	44	46	47	48	4A	4B	4C	4D	25	FF	FF	E6	57	58	43	56
48-63	42	4E	3F	2E	2F	15	FF	2A	FF	20	FF	50	51	52	53	54
64-79	55	56	57	58	59	FF	FF	37	38	39	2D	34	35	36	2B	31
80-95	32	33	30	2E	FF	FF	3E	5A	5B	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

*

Lower Case Definitions

0-15	FF	1B	26	82	22	27	28	2D	8A	5F	87	85	29	3D	08	09
16-31	61	7A	65	72	74	79	75	69	6F	70	FB	24	0D	FF	71	73
32-47	64	66	67	68	6A	6B	6C	6D	97	FF	FF	2A	77	78	63	76
48-63	62	6E	2C	3B	3A	21	FF	2A	FF	20	FF	40	41	42	43	44
64-79	45	46	47	48	49	FF	FF	FF	1A	FF	2D	15	FF	06	2B	FF
80-95	0A	FF	FF	FF	FF	FF	3C	4A	4B	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

*

Control Key Definitions

0-15	FF	1B	FF	FF	FF	FF	1B	FF	FF	1C	1E	00	1D	FF	7F	FF
16-31	01	1A	05	12	14	19	15	09	0F	10	FF	FF	0A	FF	11	13
32-47	04	06	07	08	0A	0B	0C	0D	FF	FF	FF	FF	17	18	03	16
48-63	02	0E	FF	FF	FF	FF	FF	FF	FF	20	FF	20	21	22	23	24
64-79	25	26	27	28	29	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
80-95	FF	FF	FF	FF	FF	FF	FF	2A	2B	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

*

Alternate Key Definitions

0-15	FF	FF	FF	7E	23	7B	5B	7C	60	5C	5E	40	5D	7D	FF	FF
16-31	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	0F	FF	FF	FF	FF
32-47	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
48-63	FF	FF	FF	FF	FF	FF	FF	FF	FF	20	FF	30	31	32	33	34
64-79	35	36	37	38	39	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
80-95	FF	FF	FF	FF	FF	FF	FF	3A	3B	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

```
*  Define GERMAN 84-key keyboard
*
*  Note: X'FF' means ignore this keystroke

*
```

Scan Codes	Upper Case Definitions
0-15	|FF|1B|21|22|15|24|25|26|2F|28|29|3D|3F|FC|08|FF|
16-31	|51|57|45|52|54|5A|55|49|4F|50|9A|2A|0D|FF|41|53|
32-47	|44|46|47|48|4A|4B|4C|99|8E|3E|FF|5E|59|58|43|56|
48-63	|42|4E|4D|3B|3A|5F|FF|FF|FF|20|FF|50|51|52|53|54|
64-79	|55|56|57|58|59|FF|FF|37|38|39|2D|34|35|36|2B|31|
80-95	|32|33|30|2E|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
96-111	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
112-127	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

*

Lower Case Definitions	
0-15	|FF|1B|31|32|33|34|35|36|37|38|39|30|E1|FD|08|09|
16-31	|71|77|65|72|74|7A|75|69|6F|70|81|2B|0D|FF|61|73|
32-47	|64|66|67|68|6A|6B|6C|94|84|3C|FF|23|79|78|63|76|
48-63	|62|6E|6D|2C|2E|2D|FF|2A|FF|20|FF|40|41|42|43|44|
64-79	|45|46|47|48|49|FF|FF|FF|1A|FF|2D|15|FF|06|2B|FF|
80-95	|0A|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
96-111	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
112-127	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

*

Control Key Definitions	
0-15	|FF|1B|FF|00|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|7F|FF|
16-31	|11|17|05|12|14|1A|15|09|0F|10|1B|1D|0A|FF|01|13|
32-47	|04|06|07|08|0A|0B|0C|FF|FF|1C|FF|1E|19|18|03|16|
48-63	|02|0E|0D|FF|FF|1F|FF|FF|FF|20|FF|20|21|22|23|24|
64-79	|25|26|27|28|29|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
80-95	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
96-111	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
112-127	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

*

Alternate Key Definitions	
0-15	|FF|FF|FF|40|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
16-31	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|5B|5D|FF|FF|FF|FF|
32-47	|FF|FF|FF|FF|FF|FF|FF|FF|FF|5C|FF|FF|FF|FF|FF|FF|
48-63	|FF|FF|FF|FF|FF|FF|FF|FF|FF|20|FF|30|31|32|33|34|
64-79	|35|36|37|38|39|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
80-95	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
96-111	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
112-127	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

```
*  Define GERMAN 102-key keyboard
*
*  Note: X'FF' means ignore this keystroke

*
```

Scan Codes — Upper Case Definitions

Scan Codes															
0-15	FF	1B	21	22	15	24	25	26	2F	28	29	3D	3F	FC	08	FF
16-31	51	57	45	52	54	5A	55	49	4F	50	9A	2A	0D	FF	41	53
32-47	44	46	47	48	4A	4B	4C	99	8E	FA	FF	27	59	58	43	56
48-63	42	4E	4D	3B	3A	5F	FF	2A	FF	20	FF	50	51	52	53	54
64-79	55	56	57	58	59	FF	FF	37	38	39	2D	34	35	36	2B	31
80-95	32	33	30	2E	FF	FF	3E	5A	5B	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

*

Lower Case Definitions

0-15	FF	1B	31	32	33	34	35	36	37	38	39	30	E1	FD	08	09
16-31	71	77	65	72	74	7A	75	69	6F	70	81	2B	0D	FF	61	73
32-47	64	66	67	68	6A	6B	6C	94	84	5E	FF	23	79	78	63	76
48-63	62	6E	6D	2C	2E	2D	FF	2A	FF	20	FF	40	41	42	43	44
64-79	45	46	47	48	49	FF	FF	FF	1A	FF	2D	15	FF	06	2B	FF
80-95	0A	FF	FF	FF	FF	FF	3C	4A	4B	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

*

Control Key Definitions

0-15	FF	1B	FF	FF	FF	FF	FF	FF	FF	1B	1D	FF	1C	FF	7F	FF
16-31	11	17	05	12	14	1A	15	09	0F	10	FF	FF	0A	FF	01	13
32-47	04	06	07	08	0A	0B	0C	FF	FF	1E	FF	FF	19	18	03	16
48-63	02	0E	0D	FF	FF	1F	FF	FF	FF	20	FF	20	21	22	23	24
64-79	25	26	27	28	29	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
80-95	FF	FF	FF	FF	FF	FF	FF	2A	2B	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

*

Alternate Key Definitions

0-15	FF	FF	FF	FF	FF	FF	FF	FF	7B	5B	5D	7D	5C	FF	FF	FF
16-31	40	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	7E	FF	FF	FF	FF
32-47	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
48-63	FF	FF	E6	FF	FF	FF	FF	FF	FF	20	FF	30	31	32	33	34
64-79	35	36	37	38	39	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
80-95	FF	FF	FF	FF	FF	FF	7C	3A	3B	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

```
*  Define ITALIAN 84-key keyboard
*
*  Note: X'FF' means ignore this keystroke

*
Scan Codes                Upper Case Definitions

     0-15      |FF|1B|21|22|9C|24|25|26|2F|28|29|3D|3F|5E|08|FF|
    16-31      |51|57|45|52|54|59|55|49|4F|50|82|2A|0D|FF|41|53|
    32-47      |44|46|47|48|4A|4B|4C|40|23|3E|FF|15|5A|58|43|56|
    48-63      |42|4E|4D|3B|3A|5F|FF|FF|FF|20|FF|50|51|52|53|54|
    64-79      |55|56|57|58|59|FF|FF|37|38|39|2D|34|35|36|2B|31|
    80-95      |32|33|30|2E|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
    96-111     |FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
   112-127     |FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

*
                          Lower Case Definitions

     0-15      |FF|1B|31|32|33|34|35|36|37|38|39|30|27|8D|08|09|
    16-31      |71|77|65|72|74|79|75|69|6F|70|8A|2B|0D|FF|61|73|
    32-47      |64|66|67|68|6A|6B|6C|95|85|3C|FF|97|7A|78|63|76|
    48-63      |62|6E|6D|2C|2E|2D|FF|2A|FF|20|FF|40|41|42|43|44|
    64-79      |45|46|47|48|49|FF|FF|FF|1A|FF|2D|15|FF|06|2B|FF|
    80-95      |0A|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
    96-111     |FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
   112-127     |FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

*
                          Control Key Definitions

     0-15      |FF|1B|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|1E|7F|FF|
    16-31      |11|17|05|12|14|19|15|09|0F|10|1B|1D|0A|FF|01|13|
    32-47      |04|06|07|08|0A|0B|0C|00|FF|1C|FF|FF|1A|18|03|16|
    48-63      |02|0E|0D|FF|FF|1F|FF|FF|FF|20|FF|20|21|22|23|24|
    64-79      |25|26|27|28|29|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
    80-95      |FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
    96-111     |FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
   112-127     |FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

*
*  Alternate Key Definitions

     0-15      |FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
    16-31      |FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|5B|5D|FF|FF|FF|FF|
    32-47      |FF|FF|FF|FF|FF|FF|FF|FF|FF|5C|FF|FF|FF|FF|FF|FF|
    48-63      |FF|FF|FF|FF|FF|FF|FF|FF|FF|20|FF|30|31|32|33|34|
    64-79      |35|36|37|38|39|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
    80-95      |FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
    96-111     |FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
   112-127     |FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
```

```
*  Define ITALIAN 102-key keyboard
*
*  Note: X'FF' means ignore this keystroke

*
```

Scan Codes — Upper Case Definitions

Scan Codes															
0-15	FF	1B	21	22	9C	24	25	26	2F	28	29	3D	3F	5E	08	FF
16-31	51	57	45	52	54	59	55	49	4F	50	82	2A	0D	FF	41	53
32-47	44	46	47	48	4A	4B	4C	87	A2	7C	FF	15	5A	58	43	56
48-63	42	4E	4D	3B	3A	5F	FF	2A	FF	20	FF	50	51	52	53	54
64-79	55	56	57	58	59	FF	FF	37	38	39	2D	34	35	36	2B	31
80-95	32	33	30	2E	FF	FF	3E	5A	5B	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

*

Lower Case Definitions

0-15	FF	1B	31	32	33	34	35	36	37	38	39	30	27	8D	08	09
16-31	71	77	65	72	74	79	75	69	6F	70	8A	2B	0D	FF	61	73
32-47	64	66	67	68	6A	6B	6C	95	85	5C	FF	97	7A	78	63	76
48-63	62	6E	6D	2C	2E	2D	FF	2A	FF	20	FF	40	41	42	43	44
64-79	45	46	47	48	49	FF	FF	FF	1A	FF	2D	15	FF	06	2B	FF
80-95	0A	FF	FF	FF	FF	FF	3C	4A	4B	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

*

Control Key Definitions

0-15	FF	1B	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	1E	7F	FF
16-31	11	17	05	12	14	19	15	09	0F	10	1B	1D	0A	FF	01	13
32-47	04	06	07	08	0A	0B	0C	00	FF	1C	FF	FF	1A	18	03	16
48-63	02	0E	0D	FF	FF	1F	FF	FF	FF	20	FF	20	21	22	23	24
64-79	25	26	27	28	29	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
80-95	FF	FF	FF	FF	FF	FF	FF	2A	2B	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

*

Alternate Key Definitions

0-15	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
16-31	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	5B	5D	FF	FF	FF	FF
32-47	FF	FF	FF	FF	FF	FF	FF	40	23	FF	FF	FF	FF	FF	FF	FF
48-63	FF	FF	FF	FF	FF	FF	FF	FF	FF	20	FF	30	31	32	33	34
64-79	35	36	37	38	39	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
80-95	FF	FF	FF	FF	FF	FF	FF	3A	3B	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

```
*  Define SPANISH 84-key keyboard
*
*  Note: X'FF' means ignore this keystroke

*
```

Scan Codes — Upper Case Definitions

Scan Codes															
0-15	FF	1B	AD	A8	23	24	25	2F	26	2A	28	29	5F	2B	08	FF
16-31	51	57	45	52	54	59	55	49	4F	50	FE	FB	0D	FF	41	53
32-47	44	46	47	48	4A	4B	4C	A5	3A	3E	FF	80	5A	58	43	56
48-63	42	4E	4D	3F	21	22	FF	FF	FF	20	FF	50	51	52	53	54
64-79	55	56	57	58	59	FF	FF	37	38	39	2D	34	35	36	2B	31
80-95	32	33	30	2E	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

```
*
```

Lower Case Definitions

0-15	FF	1B	31	32	33	34	35	36	37	38	39	30	2D	3D	08	09
16-31	71	77	65	72	74	79	75	69	6F	70	FD	FC	0D	FF	61	73
32-47	64	66	67	68	6A	6B	6C	A4	3B	3C	FF	87	7A	78	63	76
48-63	62	6E	6D	2C	2E	27	FF	2A	FF	20	FF	40	41	42	43	44
64-79	45	46	47	48	49	FF	FF	FF	1A	FF	2D	15	FF	06	2B	FF
80-95	0A	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

```
*
```

Control Key Definitions

0-15	FF	1B	FF	00	FF	FF	FF	FF	FF	FF	FF	FF	1F	FF	7F	FF
16-31	11	17	05	12	14	19	15	09	0F	10	1B	1D	0A	FF	01	13
32-47	04	06	07	08	0A	0B	0C	FF	FF	1C	FF	FF	1A	18	03	16
48-63	02	0E	0D	FF	FF	FF	FF	FF	FF	20	FF	20	21	22	23	24
64-79	25	26	27	28	29	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
80-95	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

```
*
```

Alternate Key Definitions

0-15	FF	FF	FF	40	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
16-31	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	5B	5D	FF	FF	FF	FF
32-47	FF	FF	FF	FF	FF	FF	FF	FF	FF	5C	FF	FF	FF	FF	FF	FF
48-63	FF	FF	FF	FF	FF	FF	FF	FF	FF	20	FF	30	31	32	33	34
64-79	35	36	37	38	39	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
80-95	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

```
*  Define SPANISH 102-key keyboard
*
*  Note: X'FF' means ignore this keystroke

*
```

Scan Codes	Upper Case Definitions
0-15	|FF|1B|21|22|60|24|25|26|2F|28|29|3D|3F|A8|08|FF|
16-31	|51|57|45|52|54|59|55|49|4F|50|FB|2A|0D|FF|41|53|
32-47	|44|46|47|48|4A|4B|4C|A5|FE|A6|FF|80|5A|58|43|56|
48-63	|42|4E|4D|3B|3A|5F|FF|2A|FF|20|FF|50|51|52|53|54|
64-79	|55|56|57|58|59|FF|FF|37|38|39|2D|34|35|36|2B|31|
80-95	|32|33|30|2E|FF|FF|3E|5A|5B|FF|FF|FF|FF|FF|FF|FF|
96-111	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
112-127	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

*

	Lower Case Definitions
0-15	|FF|1B|31|32|33|34|35|36|37|38|39|30|27|AD|08|09|
16-31	|71|77|65|72|74|79|75|69|6F|70|FC|2B|0D|FF|61|73|
32-47	|64|66|67|68|6A|6B|6C|A4|FD|A7|FF|87|7A|78|63|76|
48-63	|62|6E|6D|2C|2E|2D|FF|2A|FF|20|FF|40|41|42|43|44|
64-79	|45|46|47|48|49|FF|FF|FF|1A|FF|2D|15|FF|06|2B|FF|
80-95	|0A|FF|FF|FF|FF|FF|3C|4A|4B|FF|FF|FF|FF|FF|FF|FF|
96-111	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
112-127	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

*

	Control Key Definitions
0-15	|FF|1B|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|7F|FF|
16-31	|11|17|05|12|14|19|15|09|0F|10|1B|1D|0A|FF|01|13|
32-47	|04|06|07|08|0A|0B|0C|FF|FF|1C|FF|FF|1A|18|03|16|
48-63	|02|0E|0D|FF|FF|1F|FF|FF|FF|20|FF|20|21|22|23|24|
64-79	|25|26|27|28|29|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
80-95	|FF|FF|FF|FF|FF|FF|FF|2A|2B|FF|FF|FF|FF|FF|FF|FF|
96-111	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
112-127	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

*

	Alternate Key Definitions
0-15	|FF|FF|7C|40|23|FF|FF|AA|FF|FF|FF|FF|FF|FF|FF|FF|
16-31	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|5B|5D|FF|FF|FF|FF|
32-47	|FF|FF|FF|FF|FF|FF|FF|FF|7B|5C|FF|7D|FF|FF|FF|FF|
48-63	|FF|FF|FF|FF|FF|FF|FF|FF|FF|20|FF|30|31|32|33|34|
64-79	|35|36|37|38|39|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
80-95	|FF|FF|FF|FF|FF|FF|FF|3A|3B|FF|FF|FF|FF|FF|FF|FF|
96-111	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
112-127	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

```
*  Define USA 84-key keyboard
*
*  Note: X'FF' means ignore this keystroke

*
```

Scan Codes	Upper Case Definitions														
0-15	FF	1B	21	40	23	24	25	5E	26	2A	28	29	5F	2B	08	FF
16-31	51	57	45	52	54	59	55	49	4F	50	7B	7D	0D	FF	41	53
32-47	44	46	47	48	4A	4B	4C	3A	22	7E	FF	7C	5A	58	43	56
48-63	42	4E	4D	3C	3E	3F	FF	FF	FF	20	FF	50	51	52	53	54
64-79	55	56	57	58	59	FF	FF	37	38	39	2D	34	35	36	2B	31
80-95	32	33	30	2E	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

```
*
```

Lower Case Definitions															
0-15	FF	1B	31	32	33	34	35	36	37	38	39	30	2D	3D	08	09
16-31	71	77	65	72	74	79	75	69	6F	70	5B	5D	0D	FF	61	73
32-47	64	66	67	68	6A	6B	6C	3B	27	60	FF	5C	7A	78	63	76
48-63	62	6E	6D	2C	2E	2F	FF	2A	FF	20	FF	40	41	42	43	44
64-79	45	46	47	48	49	FF	FF	FF	1A	FF	2D	15	FF	06	2B	FF
80-95	0A	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

```
*
```

Control Key Definitions															
0-15	FF	1B	FF	00	FF	FF	FF	1E	FF	FF	FF	FF	1F	FF	7F	FF
16-31	11	17	05	12	14	19	15	09	0F	10	1B	1D	0A	FF	01	13
32-47	04	06	07	08	0A	0B	0C	FF	FF	FF	FF	1C	1A	18	03	16
48-63	02	0E	0D	FF	FF	FF	FF	FF	FF	20	FF	20	21	22	23	24
64-79	25	26	27	28	29	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
80-95	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

```
*
```

Alternate Key Definitions															
0-15	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
16-31	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
32-47	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
48-63	FF	FF	FF	FF	FF	FF	FF	FF	FF	20	FF	30	31	32	33	34
64-79	35	36	37	38	39	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
80-95	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
96-111	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
112-127	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

```
*  Define USA 101-key keyboard
*
*  Note: X'FF' means ignore this keystroke

*
```

Scan Codes	Upper Case Definitions
0-15	|FF|1B|21|40|23|24|25|5E|26|2A|28|29|5F|2B|08|FF|
16-31	|51|57|45|52|54|59|55|49|4F|50|7B|7D|0D|FF|41|53|
32-47	|44|46|47|48|4A|4B|4C|3A|22|7E|FF|7C|5A|58|43|56|
48-63	|42|4E|4D|3C|3E|3F|FF|FF|FF|20|FF|50|51|52|53|54|
64-79	|55|56|57|58|59|FF|FF|37|38|39|2D|34|35|36|2B|31|
80-95	|32|33|30|2E|FF|FF|FF|5A|5B|FF|FF|FF|FF|FF|FF|FF|
96-111	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
112-127	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

*

Lower Case Definitions	
0-15	|FF|1B|31|32|33|34|35|36|37|38|39|30|2D|3D|08|09|
16-31	|71|77|65|72|74|79|75|69|6F|70|5B|5D|0D|FF|61|73|
32-47	|64|66|67|68|6A|6B|6C|3B|27|60|FF|5C|7A|78|63|76|
48-63	|62|6E|6D|2C|2E|2F|FF|2A|FF|20|FF|40|41|42|43|44|
64-79	|45|46|47|48|49|FF|FF|FF|1A|FF|2D|15|FF|06|2B|FF|
80-95	|0A|FF|FF|FF|FF|FF|FF|4A|4B|FF|FF|FF|FF|FF|FF|FF|
96-111	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
112-127	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

*

Control Key Definitions	
0-15	|FF|1B|FF|00|FF|FF|FF|1E|FF|FF|FF|FF|1F|FF|7F|FF|
16-31	|11|17|05|12|14|19|15|09|0F|10|1B|1D|0A|FF|01|13|
32-47	|04|06|07|08|0A|0B|0C|FF|FF|FF|FF|1C|1A|18|03|16|
48-63	|02|0E|0D|FF|FF|FF|FF|FF|FF|20|FF|20|21|22|23|24|
64-79	|25|26|27|28|29|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
80-95	|FF|FF|FF|FF|FF|FF|FF|2A|2B|FF|FF|FF|FF|FF|FF|FF|
96-111	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
112-127	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

*

Alternate Key Definitions	
0-15	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
16-31	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
32-47	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
48-63	|FF|FF|FF|FF|FF|FF|FF|FF|FF|20|FF|30|31|32|33|34|
64-79	|35|36|37|38|39|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
80-95	|FF|FF|FF|FF|FF|FF|FF|3A|3B|FF|FF|FF|FF|FF|FF|FF|
96-111	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|
112-127	|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|FF|

Appendix B

DEFINING FUNCTION KEYS

The function keys can be defined in two ways. They can be part of the overall keyboard definition item in the KEYBOARD file, or they can be defined as items in the FUNCKEYS file. The definitions in the KEYBOARD file can be selected by using the SET-KBRD verb; the definitions in the FUNCKEYS file can be selected by using the SET-FUNC verb. These definitions affect line 0 (memory-mapped monitor) only.

The default function keys are based on the USA 101-key keyboard as it is shipped. The actual default function keys are coded into the PICK operating system and cannot be modified. The default function keys are loaded each time the system is booted.

Each item in the FUNCKEYS file consists of four attributes with 12 hexadecimal values each. Attribute 1 contains the values to be transmitted when the function key is pressed with the <SHIFT> key; attribute 2 contains the values to be transmitted when the function key is pressed by itself; attribute 3 contains the values to be transmitted when the function key is pressed with the <CTRL> key; attribute 4 contains the valued to be transmitted when the function key is pressed with the <ALT> key.

Within each attribute, the first value corresponds to function key 1, the second value corresponds to function key 2, and so on.

NOTE: The function keys generate scan codes 59-68 and 87-88. (For information about scan codes, see Appendix A.)

Three items are currently included in the FUNCKEYS file: DEFAULT, TEST, and NULL. The function key definitions in DEFAULT are the same as those that are loaded when the system is booted; the function key definitions in TEST are for demonstrations; the function key definitions in NULL can be used to clear all function key definitions.

Figure B-1 displayes the DEFAULT item, Figure B-2 displays the TEST item, and Figure B-3 displays the NULL item.

NOTE: Function keys echo two characters. The first character is STX (ASCII character 02) and cannot be changed; the second is the character in the KEYBOARD or FUNCKEYS item and can be changed.

```
    DEFAULT
001 *
002 * Define default function keys
003 *
004 505152535455565758595A5B      function key + shift
005 4041424344454647484 94A4B     function key only
006 2021222324252627282 92A2B     function key + ctrl
007 3031323334353637383 93A3B     function key + alt
008 *
```

Figure 1 DEFAULT Item in FUNCKEYS File

```
    TEST
001 *
002 * Define function keys for testing
003 *
004 4142434445464748494A4B4C      function key + shift
005 515253545556575859 5A5B5C     function key only
006 212223242526272829 2A2B2C     function key + ctrl
007 313233343536373839 3A3B3C     function key + alt
008 *
```

Figure 2 TEST Item in FUNCKEYS File

```
    NULL
001 *
002 * Define function keys to null
003 *
004 FFFFFFFFFFFFFFFFFFFFFFFF      function key + shift
005 FFFFFFFFFFFFFFFFFFFFFFFF      function key only
006 FFFFFFFFFFFFFFFFFFFFFFFF      function key + ctrl
007 FFFFFFFFFFFFFFFFFFFFFFFF      function key + alt
008 *
```

Figure 3 NULL Item in FUNCKEYS File

, 8-19
*, 3-11
+, 5-37
-, 5-37, 9-15
A - modifier, 6-54
A command, 5-17
A correlatives, 6-125
A indicator, 7-62, 7-63
A option, 7-53, 7-55, 7-61, 7-111
A-items, 2-7, 2-26, 2-28, 6-10
ABORT statement, 9-41
Aborted, 7-62
ABS, 10-56
ABS EXTENSION, 12-11, 11-27
ABS frames, 10-9
ABS function, 9-42
ABS load, 10-41
ABS Restore, 12-10
ACC File
  TERM-TYPE, 3-27, 10-12, 10-23, 10-25, 10-27
ACCESS - input conversions, 6-34
ACCESS attribute names, 6-7
ACCESS delimiters, 6-27
ACCESS file-names, 6-7
ACCESS item-ids, 6-7
ACCESS item-lists, 6-7
ACCESS modifiers, 6-7
ACCESS options, 6-7, 6-55
ACCESS output specifications, 6-7
ACCESS print limiters, 6-7
ACCESS selection criteria, 6-7
ACCESS sentence, 6-9
ACCESS statements - formation, 6-7, 6-9
ACCESS verbs, 6-7, 6-17
Accessing a file, 2-5, 9-130
Accessing files in other accounts, 2-22
Accessing item-id, 9-131
Accessing multi-values, 6-72
Accessing single attributes, 9-136
Account name: print file, 7-62
Account specification, 2-25
ACCOUNT-RESTORE, 10-51
ACCOUNT-SAVE, 10-51
Accounting file, 10-23, 10-25
Accounting history item, 10-23, 10-25, 10-27
Accounting history update, 10-19
accountname example, 7-67
accountname option, 7-25
accountname option in SP-EDIT, 7-30
accountname specification, 7-61
ACTIVE, 7-68
Active user item, 10-23
Additional workspace, 10-6
AGAIN command - Editor, 4-14
ALIGN, 7-35, 7-41, 7-43, 7-62
All items, 3-11
ALPHA function, 9-43
Alternate dictionary definition, 6-13
AM, 2-15
AMC, 2-26
Amc of 9998 or 9999, 6-10
Ampersand in RUNOFF, 8-19
AN - modifier, 6-54
AND clause, 6-43
AND clause and selection, 6-43
AND clauses, 6-41
AND connective, 6-19, 6-24, 6-41
ANDed selection criteria, 6-43
ANDed value phrases, 6-41
ANY - modifier, 6-54
ARE - modifier, 6-54
Arithmetic capability, 5-37
Arithmetic functions, 6-125
Arithmetic operators, 9-25
Array passing, 9-48
Arrays, 9-13, 9-67
Arrays (dynamic) - Deletion, 9-66
Arrays (dynamic) - Extraction, 9-78
Arrays (dynamic) - Insertion, 9-96
Arrays (dynamic) - Replacement, 9-139
Arrays - Dynamic, 9-13
Arrays - Dynamic, LOCATE, 9-100
AS command - Editor, 4-33
ASCII - codes, 9-194
ASCII Codes, 12-27
ASCII conversions, 6-116
ASCII function, 9-44
Assembly Formatting, 4-33
Assigning values to variables, 9-45
Assignment - Array, 9-106
Assignment indicators, 7-73
Assignment interrogation, 7-19, 7-20, 7-21, 7-73
Assignment specification, 7-15
Assignment statement, 9-45
Assignment status, 7-73
Attribute 7, 6-34
Attribute 7 in selection, 6-30
Attribute definition items, 2-7, 2-26, 6-10
Attribute definitions, 6-10
Attribute mark count, 2-26
Attribute names, 2-7, 2-26, 6-10
Attribute values - breaking on, 6-61
Attributes, 2-3, 2-15
Attributes - Accessing, 9-163
Attributes - controlling, 6-97, 6-99

Attributes - dependent, 6-97, 6-99
Attributes - Updating, 9-163
Automatic execution of a PROC at LOGON, 3-7
Available, 7-62
Available print file control record, 7-63
Available space, 10-7, 10-34
B, 5-15
B - RUNOFF Command, 8-5
B -debug command, 9-170
B option, 7-49, 7-53, 7-55, 7-57, 7-68
'B' option, 6-56, 6-64
Backspacing, 3-3
Backup, 10-47
BASE, 2-11
BASIC
  Format Masks, 9-93
  Numeric Masks, 9-93
BASIC - IF statement, 9-92
BASIC compiler error messages, 9-190
BASIC compiler options, 9-17, 9-19
BASIC debugger, 9-20, 9-165
BASIC debugger - Breakpoint Table, 9-170
BASIC debugger - Changing a variable, 9-172
BASIC debugger - Commands, 9-197
BASIC debugger - Execution control, 9-171
BASIC debugger - Messages, 9-199
BASIC debugger - Trace table, 9-169
BASIC debugger - usage, 9-167
BASIC file structure, 9-11
BASIC intrinsic function summary, 9-188
BASIC language definition, 9-5
Basic Output Formatting, 9-124
BASIC program - Cataloging, 9-21
BASIC program - Debugging, 9-165, 9-167, 9-169, 9-171
BASIC program - Decataloging, 9-21
BASIC program - Executing, 9-20
BASIC program - Running, 9-20
BASIC program - Sharing, 9-21
BASIC PROGRAM FILE STRUCTURE, 2-31
BASIC run-time error messages, 9-192
BASIC symbol tables items, 9-165
BASIC syntax summary, 9-184
Baud Rate, 12-26
BEGIN PAGE - RUNOFF Command, 8-5
Being Output, 7-62
.bf, 8-19
BLANK as delimiter, 6-15
Blanks - PICK/BASIC, 9-12
BLOCK-CONVERT, 10-14
BLOCK-CONVERT file, 10-12
BLOCK-PRINT, 3-13
BMS of a file, 2-11
BO, 5-15
Boldface, 8-20
BOLDFACE in RUNOFF, 8-19
Boolean expressions, 9-33
Boolean functions, 9-113
Booting Your System, 12-4
BOTTOM command - Editor, 4-12
BOX - RUNOFF Command, 8-5
BOX OFF - RUNOFF Command, 8-5
BP - RUNOFF Command, 8-5
BP and line skipping, 8-5
BP from SP-EDIT, 7-39
Branch, 9-88, 9-117
Branch - Conditional, 9-49
BREAK - RUNOFF Command, 8-5
BREAK OFF statement, 9-46
BREAK ON statement, 9-46
BREAK-ON modifier, 6-63
Breaking on attribute values, 6-61
Breakpoint Table - PICK/BASIC debugger, 9-170
Buffer pointers, 5-15
Buffers in the Editor, 4-5
Buffers, PROC, 5-8
BY modifier, 6-69
BY-DSND modifier, 6-69
BY-EXP modifier, 6-72
BY-EXP-DSND modifier, 6-72
C, 5-37
C - code, 6-108
C - output option, 6-50
C - RUNOFF Command, 8-6
C Command - Editor, 4-36
C definition code, 6-99
C indicator, 7-62, 7-63, 7-73
C option, 7-61, 7-81
C specification, 7-17, 7-21
'C' option, 6-56
CALL statement, 9-47
Calling a subroutine, 5-43
Calling another PROC, 5-41
Capitalize sentences - RUNOFF Command, 8-5
CASE statement, 9-49
CATALOGed programs, change in storage, 9-11
Cataloging PICK/BASIC programs, 9-21
CENTER - RUNOFF Command, 8-6
CHAIN - RUNOFF Command, 8-6
CHAIN from SP-EDIT, 7-39
CHAIN statement, 9-50
Changes: SP-ASSIGN, 7-15

Changing copy count, 7-28
Changing form number, 7-28
Changing output queue, 7-28
Changing the form number, 7-37
Changing the output queue specification, 7-37
CHAPTER command in RUNOFF, 8-7
CHAR function, 9-52
[character, 6-37
Character manipulation, 6-130
CHARGE-TO, 3-15
CHARGES, 3-15
CHECK-SUM, 10-33
Choked output, 7-17
Choked output example, 7-23
Choked print file generation, 7-17
Clear screen, 9-38
CLEAR statement, 9-53
CLEAR-FILE, 2-32, 2-35
CLEARFILE statement, 9-54
Clearing a file, 2-32, 2-35
Clearing ACC file, 10-27
Clearing buffers, 5-25
Clearing variable values, 9-53
CLOCK Verb, 3-16
Close print file, 7-15
Closed, 7-62
Coding techniques, 9-174
COL-HDR-SUPP modifier, 6-50
COL1()/COL2() function, 9-55
Cold-start, 10-41
Coldstarts, 7-81
COLON - EDITOR, 4-9
Color, 12-21
Columnar format, 6-45
Columnar Positions - Editor, 4-36
.* Command, 8-7
? Command - Editor, 4-35, 4-36
Commands - PICK/BASIC debugger, 9-197
Comment lines, 5-37
Comments in RUNOFF, 8-7
COMMON statement, 9-56
Common variables, 9-56
Compiler error messages, 9-190
Compiling a PICK/BASIC program, 9-15
Complex item-lists, 6-21, 6-26
Computed branch, 9-117
Concatenating print files, 7-17, 7-111
Concatenation, 6-108, 9-27
Conditional branch, 9-92, 9-94
Conditional execution, 5-29
Conditional values, 6-37
Configuration, 10-53
Configuration Extension, 12-12
Connectives, 6-24
Connectives - AND, 6-19
Connectives - logical, 6-19
Connectives - OR, 6-19
Constants, 9-23
CONTENTS command in RUNOFF, 8-7
Continuous output to the terminal, 7-38
Control - breaks, 6-63
Control Characters in RUNOFF, 8-19
Control-H, 3-3
Control-R, 3-3
Control-W, 3-3
Control-X, 3-3
Controlling attributes, 6-97, 6-99
Controlling definition code, 6-99
CONVERSION codes, 6-101
Conversion operator, F - code, 6-117
Conversions - ASCII, 6-116
Conversions - user, 6-116
CONVERSIONS w/sort keys, 6-70
Copy - Array, 9-106
Copy count, 7-15, 7-19
Copy count change, 7-28, 7-31
Copy count specification, 7-62, 7-73
Copy processor, 2-37, 2-38
COPY processor options, 2-40
COPY-LIST verb, 6-88
COPYDOS, 12-41, 11-20
Copying a select-list, 6-88
Copying data, 2-37, 2-38
Copying data to the terminal or line-printer, 2-40
Copying file to file - REFORMAT & SREFORMAT, 6-79
CORRELATIVE codes, 6-101
Correlatives - Repeat for multivalues, 6-120
CORRELATIVES w/ sort keys, 6-70
COS function, 9-58
Count field, 10-31
Count field of an item, 2-15, 6-10
Count field size, 10-31
Count field storage form, 10-31
COUNT function, 9-59
COUNT verb, 6-81
CREATE-ACCOUNT, 10-19, 10-35
CREATE-FILE, 2-32, 2-33
Creating a file, 2-32, 2-33
Creating a new account, 10-35
Creating a PICK/BASIC program, 9-15
Creating Null Lines - Editor, 4-26
Creating', 9-15
CRT command in RUNOFF, 8-7
CRT Statement, 9-60
CS - RUNOFF Command, 8-5
<CTRL> characters - special, 3-4

Current buffer, PROC, 5-8
Current Date, 3-19
Current line concept, 4-5
Current Line? - Editor, 4-35
Cursor addressing, 3-25
Cursor control, 5-23, 9-38
D, 5-21
D - code, 6-110
D -debug command, 9-170
D definition code, 6-99
D option, 7-25, 7-28, 7-53, 7-57
D response, 7-36
'D' option, 6-56, 6-64
D-items, 2-7, 2-20, 2-28
Data -- selection by, 6-30
Data evaluation, 6-30
Data existance -- algorithm, 6-31
Data existence, 6-31
Data file as dictionary, 6-13
Data format errors in files, 10-31
Data input, 4-18, 5-19
Data Length, 12-26
Data movement, PROC, 5-8
Data representation, 9-23
Data restore, 10-42, 10-44
DATA statement, 9-61
Data-level files: multiple, 2-25
Date
  Setting, 3-16, 3-19, 12-20
Date conversion - input, 6-34
Date format, 6-110
  Default, 12-20
  Standard, 12-20
DATE() function, 9-63
DCD-OFF, 12-24
DCD-ON, 12-24
DCOUNT function, 9-64
$ -debug command, 9-173
Debugger, 3-31
Debugging PICK/BASIC programs, 9-165, 9-167, 9-169
Decataloging PICK/BASIC programs, 9-21
Default output specification, 6-49
Default print file to tape record length, 7-37
Default relational connective, 6-28
Default tape record size, 7-97
Default: SP-ASSIGN, 7-22
Defining a new file, 2-33
Delete column specifications, 4-21
DELETE command - Editor, 4-21
DELETE function, 9-66
DELETE statement, 9-65
DELETE-ACCOUNT, 10-36
DELETE-FILE, 2-32, 2-35, 2-36
DELETE-LIST verb, 6-86
DELETE: SP-EDIT, 7-35
Deleting a file, 2-32, 2-36
Deleting an account, 10-36
Deleting input line, 3-3
Deleting Lines, 4-21
Deletion -- forced, 7-28
Delimiters in ACCESS, 6-27
Dependent attributes, 6-97, 6-99
Dependent definition code, 6-99
DET-SUPP modifier, 6-66
Detaching a printer, 7-52, 7-57
Detaching a printer: examples, 7-57
DICT modifier, 6-67
Dictionaries, 2-3, 2-7, 6-10
Dictionaries, sharing of, 2-9
DIM statement, 9-67
Direct file access, 2-5
Disenqueuing a print file, 7-52, 7-55
Disk usage, 10-3
DISPLAY: SP-EDIT, 7-35, 7-36
Displaying a frame, 10-10
Displaying group data, 10-28
Displaying output queues, 7-67
Displaying parameters, 5-21
Displaying prestored commands, 4-39
Displaying the print file control records, 7-60
DOS TO PICK BRIDGE, 12-41, 11-20
DTX function, 9-69
Dummy amc, 6-10
DUMP, 10-10
Dynamic arrays, 9-13, 9-66, 9-139
Dynamic arrays, EXTRACT, 9-78
Dynamic arrays,INSERT, 9-96
Dynamic arrays,LOCATE, 9-100
E - debug command, 9-171
E option, 7-61
EACH connective, 6-30
EACH modifier, 6-24
EBCDIC function, 9-70
EBCDIC to ASCII: print files, 7-111
ECHO ON/OFF statement, 9-71
EDIT verb, 4-7
EDIT-LIST verb, 6-88
Editing a PICK/BASIC program, 9-15
Editing a select-list, 6-88
Editing an item, 4-7
Editing fuctions, 3-3
Editing Q-items, 2-22
Editing', 9-15
EDITOR buffers, 4-5
EDITOR command summary, 4-9
EDITOR command syntax, 4-9
EDITOR messages, 4-41

EDITOR options, 4-7
ELSE clause, 9-92
Empty group, 2-15
END -debug command, 9-173
End of group data, 2-15
ENTER statement, 9-73
Entering data, 4-18
ENTRY #, 7-25, 7-62, 7-89
ENTRY # mark for PROC, 7-89
EOI, 4-5
EQU/EQUATE statement, 9-74
ERRMSG File
  SEQ Item, 6-131
  Sort Order, 6-131
Error message numbers: STOPPTR, 7-49
Error message numbers: T-ATT, 7-99
Error messages, 10-16
Error messages - Compiler, 9-190
Error messages - Run-time, 9-192
Error messages: print file disenqueuement, 7-59
Error messages: print file termination, 7-58
Error messages: printer detachment, 7-59
Error messages: SP-KILL, 7-58
Error messages: SP-KILL D, 7-59
Error messages: SP-KILL F, 7-59
Error messages: STARTPTR, 7-46
Error messages: STOPPTR, 7-51
Error testing, 5-35
ESCAPE character, 4-37
Evaluation of data under selection, 6-30
EVERY modifier, 6-24
EX command - Editor, 4-29
Examples of programming, 9-176, 9-177, 9-178, 9-179, 9-181
EXECUTE statement, 9-75
Executing a PICK/BASIC program, 9-20
Executing a PROC, 5-6
Executing a PROC-generated statement, 5-39
Executing TCL statements from BASIC, 9-50
Execution control - PICK/BASIC debugger, 9-171
Execution interruption, 3-31
Existence test, 6-31
Exit SP-EDIT, 7-36
Exit SPOOL T, 7-38
Explicit item-ids, 6-27
Explicit item-ids and lists, 6-27
Explicit item-lists, 6-21
Exploding sort, 6-72
EXPONENTIAL function, 9-77
Expressions, 9-25
Expressions - Boolean, 9-33
Expressions - Logical, 9-33
Expressions - Relational, 9-29, 9-31
Extended Character Set, 12-27
Extents of a file, 2-11
External subroutines, 9-47, 9-48
EXTRA ABS, 11-27
EXTRACT function, 9-78
Extraction - text, 6-109
F, 5-15
F - code, 6-117
F - code operands, 6-120
F - code, P(ropagate), 6-117
F command, 4-5
F command - Editor, 4-28
F command in RUNOFF, 8-7
F option, 6-96, 7-53, 7-55, 7-61
F response, 7-35, 7-38, 7-39
'F' option, 6-56
F-code stack operations, 6-123
FC, 12-25
FD command - Editor, 4-29
FI command - Editor, 4-28
FID, 10-3, 10-9, 10-10
FIELD function, 9-80
FILE - modifier, 6-54
File access, 3-11, 9-130, 9-136
File access method, 2-5
File area, 10-7
File BMS, 2-11
File copy, 2-38
File definition items, 2-7, 2-20
File extents, 2-11
File hashing statistics, 10-30
File hierarchy, 2-3
File items, 2-5
File items structure, 9-13
File levels, 2-3
File management processors, 2-32
FILE NAME- prompt, 7-39
File pointers, 2-20
File restore, 10-41, 10-42, 10-44, 10-45, 12-9
File save, 10-49
File space on disk, 10-7
File statistics, 6-91, 6-92
File statistics report, 10-37
File structure, 10-28
File synonym definition items, 2-22, 3-20
File translation, 6-114
File-name, 2-7
FILE-SAVE, 10-47, 10-49

Files, 10-7
FILL command in RUNOFF, 8-7
floppy diskettes, 12-17
Floppy Disks
  Formatting, 12-17
Fn option group, 7-30
Fn specification, 7-19, 7-21
'Fn' option, 6-56
Fn-m option group, 7-30
FOOTING command in RUNOFF, 8-8
FOOTING statement, 9-81
FOOTINGS, 6-56
FOR - modifier, 6-54
FOR statement, 9-83
FOR...NEXT statement, 9-85
Form number, 7-19
Form number change, 7-28, 7-31, 7-37
Form number: hold file, 7-37
Form queue, 7-15
Form queue specification, 7-74
Form specification, 7-41, 7-62, 7-68
Format - columnar, 6-45
Format - date, 6-110
Format - non-columnar, 6-45
Format - time, 6-113
Format conversion, 9-44, 9-70
FORMAT ERROR- GROUP AT xxxx message, 10-31
Format masking, MR & ML, 6-128
Format of a frame, 10-9
Format of dictionary items, summary, 2-28
Format of frames, 10-10
FORMAT Verb, 12-17
Formated terminal output, 5-23
Formats of verbs, 3-12
Forming item-lists, 6-21
Forming output specifications, 6-45
Forming selection criteria, 6-24
Forming statements, 6-9
Forming value-lists, 6-24
Frame, 10-3, 10-9
Frame format, 10-9, 10-10
Frame identifier, 10-3
Frame links, 10-9, 10-10
FS command - Editor, 4-28
@ function, 9-38, 9-93
Functions - mathematical, 6-117
G - code, 6-103
G - debug command, 9-171
G indicator, 7-62, 7-63
Generating check-sums, 10-33
GET-LIST verb, 6-23, 6-86
GLOSSARY, 1-18
GO, 5-27
GOSUB statement, 9-87
GOTO command - Editor, 4-12
GO{TO} statement, 9-88
GRAND-TOTAL modifier, 6-60
GROUP, 10-28
Group extraction, 6-103
Group format errors, 10-31
Group number, 10-31
Group, definition of, 10-31
Group-format errors, 10-31
H, 5-25
H - output option, 6-50
H indicator, 7-62, 7-63, 7-73
H option, 6-93, 7-25, 7-37
H specification, 7-16, 7-21
Handling numbers, 6-128
HASH-TEST, 10-30
HASH-TEST verb, 6-92
Hashing algorithm, 2-19, 10-31
Hashing information, 10-30
HDR-SUPP, 7-37
HDR-SUPP modifier, 6-50
HEADING command in RUNOFF, 8-8
HEADING statement, 9-89
HEADINGS, 6-56
Headings - options, 6-56, 6-64
Hierarchy of logical connectives, 6-21
Hierarchy of operators, 9-25
Hierarchy, files, 2-3
HILITE command in RUNOFF, 8-9
HOLD ENTRY #, 7-25, 7-62, 7-89
Hold file -- system admissability, 7-26
Hold file -- tape labels, 7-27
Hold file creation, 7-15, 7-16
Hold file deletion, 7-25
Hold file display, 7-36
Hold file enquement, 7-37
Hold file error message number, 7-89
Hold file interrogation, 7-25
Hold file manipulation, 7-25
Hold file number, 7-62
Hold file output, 7-25, 7-37
Hold file to data file option, 7-28
Hold files -- locked, 7-26
Hold files -- Unlocked, 7-26
Hold files: tape record size, 7-38
Hold files: to tape, 7-37
Holdfile control record, 7-62
Hyphens in RUNOFF, 8-9
I - output option, 6-50
I command in RUNOFF, 8-9
I indicator, 7-62, 7-63, 7-73
I option, 6-93, 7-81
I specification, 7-17, 7-21
ICONV function, 9-91

ID-SUPP modifier, 6-50
IF, 5-29, 5-31, 5-33
IF E Command, 5-35
IF modifier, 6-24
IF S Command, 5-35
IF statement, 9-92
IF-ELSE form, 9-92
IH, 5-25
Illegal characters in item-ids, 6-16
IM command in RUNOFF, 8-9
Immediate output, 7-15, 7-17, 7-62
Implicit item-lists, 6-23
in, 8-19
IN - modifier, 6-54
In Statement, 9-96
INACTIVE, 7-68
INCLUDE statement, 9-91
INDENT in RUNOFF, 8-9
INDENT MARGIN in RUNOFF, 8-9
INDEX command in RUNOFF, 8-10
INDEX function, 9-92
INDEX printing in RUNOFF, 8-13
Indirect subroutine calls, 9-48
INITIAL ITEM- prompt, 7-39
Initial page eject suppression, 7-41
Initial system files, 2-30
Initializing a printer, 7-41
Input, 9-96
Input buffers, PROC, 5-8
INPUT command - Editor, 4-16
INPUT Command in RUNOFF, 8-10
Input conversions, 6-34
Input null Attributes, 4-23
Input null lines, 4-23
INPUT statement, 9-93
INPUTERR statement, 9-95
INPUTNULL statement, 9-95
INPUTTRAP statement, 9-95
INSERT command - Editor, 4-18
INSERT function, 9-96
Inspecting print files being output, 7-31
INT function, 9-97
Internal Date, 6-112
Interprogram transfers, 9-73
Interrupting a process, 3-31
Intrepretive execution of PROCs, 5-6
ISTAT, 10-30
ISTAT verb, 6-91
ITEM, 10-28
Item count field, 2-15
Item length count, 10-31
Item list, 3-11
Item physical storage, 2-19
Item retrieval, 10-31
Item selection, 6-43
Item sequence from SPOOL F, 7-39
Item structure, logical, 2-17
Item structure, physical, 2-15
Item-id defintion with Q-pointers, 6-14
Item-id delimiters, 6-27
Item-id selection, 6-27, 6-28
Item-id selection - tests, 6-27
Item-id selection and lists, 6-28
Item-id specification, 6-15
Item-id structure, 6-15
Item-ids, 2-3, 2-5
Item-ids at TCL, 6-15
Item-lists - complex, 6-21
Item-lists - explicit, 6-21
Item-lists - formation, 6-21
Item-lists - implicit, 6-23
Item-lists - simple, 6-21
Items, 2-3, 2-5
ITEMS - modifier, 6-54
J command in RUNOFF, 8-10
Justification, 6-39
JUSTIFY command in RUNOFF, 8-10
K -debug command, 9-170
Key, 10-31
Keyboards, 12-21
Killing a print job, 7-52, 7-53
L - code, 6-104
L -debug command, 9-173
L indicator, 7-62, 7-64
L option, 7-25, 7-26, 7-36, 7-61, 7-81
'L' grand-total option, 6-60
'L' option, 6-64
'L'option, 6-56
Labels, 5-27, 5-29
Labels - Statement, 9-12
LC Command in RUNOFF, 8-10
LEFT MARGIN in RUNOFF, 8-10
[- left-bracket, 6-26
Left-bracket ([), 6-26
LEN function, 9-98
Levels, file, 2-3
Limiting printing of multiple values, 6-47
LINE LENGTH setting in RUNOFF, 8-10
Line-printer characteristics, 3-25
LINK-WS, 10-6
Linking to another PROC, 5-41
Linking work-space, 10-6
Linking workspace, 7-81
Links in a frame, 10-10
LIST command - Editor, 4-11
LIST verb, 6-67
LIST-FILE-STATS, 10-37
LIST-ITEM verb, 6-96

LIST-LABEL verb, 6-76
LIST-PORTS, 12-28
LISTABS verb, 7-73
LISTPEQS and SP-EDIT, 7-28
LISTPEQS by output queue, 7-67
LISTPEQS display definitions, 7-62
LISTPEQS examples, 7-65
LISTPEQS verb, 7-60
LISTPTR examples, 7-69
LISTPTR verb, 7-68
Lists and explicit item-ids, 6-27
Lists and item-id selection, 6-28
Lists of selected item-ids or values, 6-84, 6-86, 6-88
Literal strings, 5-25
Load Previous Value operator, 6-122
Loading a T-DUMP tape, 6-93
LOCATE statement, 9-100
LOCK statement, 9-102
Locked hold files, 7-26
Locked print file, 7-62
Locked print files, 7-64
Logical connectives, 6-19
Logical connectives - hierarchy, 6-21
Logical expressions, 9-33
Logical functions, 9-113
Logical item structure, 2-17
Logical operators, 9-29
LOGOFF, 3-5
LOGON, 3-5
LOGON message, 10-16
LOGON PROC, 3-7
LOGTO, 3-8
LOOP statement, 9-104
Looping, 9-83, 9-85, 9-104
Looping capability, 5-37
LOWER CASE Command in RUNOFF, 8-10
Lower case control, 8-19
LOWER CASE setting in RUNOFF, 8-10
LP -debug command, 9-173
LPTR Command in RUNOFF, 8-10
LPTR modifier, 6-67
LPV operator, 6-122
M command - Editor, 4-33
M option, 7-25, 7-28
m-n option, 7-28
m-n option in SP-EDIT, 7-30
Macro Expansion, 4-33
Magnetic tape dump and load, 6-93
Mask Character conversions, 6-130
Mask Conversions, 6-128
Mask conversions - input, 6-35
Masking Data
  MS Processing Code, 6-131, 9-35, 9-93
Masking functions, 6-35
Masking print files to upper case, 7-111
Master Dictionary, 2-3
  QFILE, 3-20
MAT statement, 9-106
MATCH operator, 9-31
Matching - Pattern, 9-31
Mathematical functions, 6-117, 6-125
MATREAD statement, 9-108
MATREADU statement, 9-109, 9-110
MATWRITE statement, 9-111
MATWRITEU statement, 9-112
Max-length of zero, 6-61
Maximum FID, 10-7
Maximum item size from SPOOL F, 7-39
MC - code, 6-130
MC functions, 6-35
MD conversion, 6-35
MD option, 7-40
MD option group, 7-28
Memory, virtual, 10-3
Memory-Mapped Monitor, 12-21
MERGE command - Editor, 4-18
MERGE DEFAULTS, 4-19
MERGING items from other files, 4-19
Messages - PICK/BASIC debugger, 9-199
Messages output by the EDITOR, 4-41
Messaging other users, 3-17
MINIMAL MERGE, 4-19
ML - code, 6-128
ML conversion, 6-35
MOD function, 9-138
MODEM, 12-25
MODIFIERS - ACCESS, 6-52
MODULO, 2-11, 2-13, 10-31
Mono, 12-21
MOUNT NEXT REEL response, 10-43
MR - code, 6-128
MR conversion, 6-35
MS (Mask Sequence) Code, 6-69, 6-131
MS option, 7-37
MS option group, 7-28
MSG (TCL-I verb), 3-17
MSP option group, 7-27
MST option group, 7-27
MT - code, 6-113
Multi-line IF statements, 9-94
Multiple data file Q-pointers, 2-25
Multiple data files, 2-25
Multiple data files for a dictionary, 2-9
Multiple line headings, 6-10, 6-45
Multiple Replacements - Editor, 4-26
MX - code, 6-116
N - debug command, 9-171

N command in RUNOFF, 8-11
N indicator, 7-62, 7-63
N option, 6-96, 7-25, 7-49, 7-53, 7-55, 7-57, 7-61, 7-68
n option in SP-EDIT, 7-30
N response, 7-35, 7-36, 7-37
n specification, 7-21
'N' option, 6-64
n-m option, 7-25, 7-49, 7-53, 7-55, 7-57, 7-61, 7-68
Names of attributes, 6-10
NATURAL LOGARITHM function, 9-99
NB - operand, 6-59
NCS Command in RUNOFF, 8-10, 8-11
ND - operand, 6-59
New account creation, 10-35
NEXT command - Editor, 4-12
Next hold file, 7-36
NEXT statement, 9-83
NF command in RUNOFF, 8-11
NJ command in RUNOFF, 8-11
No Close, 7-62
NOCAPITALIZE SENTENCES in RUNOFF, 8-10, 8-11
NOFILL mode in RUNOFF, 8-11
NOJUSTIFY command in RUNOFF, 8-11
Non-columnar format, 6-45
Nopage option in SP-EDIT, 7-38
NOPAGING command in RUNOFF, 8-11
NOPARAGRAPH command in RUNOFF, 8-11
NOT connective, 6-30
NOT ENOUGH WORK SPACE, 10-6
NOT function, 9-113
NULL - EDITOR, 4-11
NULL statement, 9-114
NUM function, 9-115
Numbers - Statement, 9-12
Numeric Mask, 9-35
O, 5-21
O indicator, 7-62, 7-63, 7-73
O option, 6-93, 7-53, 7-55
O response, 10-43
O specification, 7-21
OCONV function, 9-116
OF - modifier, 6-54
OFF, 3-5
OFF LINE, 7-74
ON GOSUB statement, 9-87
ON GOTO statement, 9-117
ON LINE, 7-74
ONLY modifier, 6-50
Open print file, 7-15
Open print files, 7-17
OPEN statement, 9-118
Opening a file, 9-118
Operands - F-code, 6-120
Operations - F-code, 6-123
Operators, 9-25
Operators - Logical, 9-29
Operators - relational, 6-19, 9-29
'' option, 6-56, 8-18
Options, 3-3, 3-11
OPTIONS - ACCESS, 6-52, 6-55
Options - Compiler, 9-15, 9-17, 9-19
Options - Compiler A, C, E, L, and P, 9-17
Options - Compiler M, S, and X, 9-19
Options - heading, 6-64
Options - Headings, 6-56
Options - output control, 6-64
Options - Runtime A, D, E, I, N, P and S, 9-20
Options, COPY processor, 2-40
Options: general protocols, 7-86
Options: SP-KILL, 7-53
OR - modifier, 6-54
OR connective, 6-19, 6-24, 6-40
ORed selection criteria, 6-42
OUT Function, 9-119
Output buffers, PROC, 5-8
Output control - options, 6-64
Output of data, 9-122
Output of spool files on tape, 7-101, 7-111
Output queue change, 7-28, 7-31
Output queue links, 7-62
Output queue number, 7-19
Output queue servicing, 7-19
Output queue specification, 7-15, 7-19, 7-41, 7-62, 7-68, 7-73, 7-74
Output queue specification change, 7-37
Output specification, 7-15
Output specification - default, 6-49
Output specifications - formation, 6-45
Output to file, REFORMAT & SREFORMAT, 6-79
Output to Tape, REFORMAT & SREFORMAT, 6-79
Output to terminal, 5-21
Outputting a hold file, 7-37
Overflow space, 10-7, 10-34
Overlapped I/O, 12-12
Override tape label, 10-43
P, 5-39
P - code, 6-106
P - option, 6-67
P -debug command, 9-173
P indicator, 7-62, 7-63, 7-73
P option, 6-96, 7-27, 7-37, 7-61
P option and PROC, 7-28

'P' grand-total option, 6-60
'P' option, 6-56, 6-64
P0 command initialization, 4-38
PAGE NUMBER command in RUNOFF, 8-11
Page skip display, 7-68
Page skip specification, 7-41, 7-74
PAGE statement, 9-119
Pagination on serial printers, 7-41
PAPER LENGTH command in RUNOFF, 8-11
PARAGRAPH setting in RUNOFF, 8-11
Parallel printer, 7-41, 7-68
Parameters, PROC, 5-8
Parity, 12-26
Password, 10-19
PASSWORD Verbs, 3-18
Pattern matching, 5-33, 9-31
Pattern testing, 5-33
PC -debug command, 9-173
PD command, 4-39
peripheral storage devices, 12-16
PH, 5-39
Physical item structure, 2-15
PICK TO DOS BRIDGE, 12-45, 11-24
PICK/BASIC, 9-15, 9-93
PICK/BASIC - ABORT, 9-41
PICK/BASIC - ABS, 9-42
PICK/BASIC - ALPHA, 9-43
PICK/BASIC - ASCII, 9-44
PICK/BASIC - BREAK ON/OFF, 9-46
PICK/BASIC - CALL, 9-47
PICK/BASIC - CALL, 9-48
PICK/BASIC - CASE, 9-49
PICK/BASIC - CHAIN, 9-50
PICK/BASIC - CHAR, 9-52
PICK/BASIC - CLEARFILE, 9-54
PICK/BASIC - COL1() / COL2(), 9-55
PICK/BASIC - Comma, Colon, 9-124
PICK/BASIC - COMMON, 9-56
PICK/BASIC - COS, 9-58
PICK/BASIC - CRT, 9-60
PICK/BASIC - DATA, 9-61
PICK/BASIC - DATE(), 9-63
PICK/BASIC - DELETE, 9-65
PICK/BASIC - DIM, 9-67
PICK/BASIC - DTX, 9-69
PICK/BASIC - EBCDIC, 9-70
PICK/BASIC - ECHO ON/OFF, 9-71
PICK/BASIC - END, 9-72
PICK/BASIC - ENTER, 9-73
PICK/BASIC - EQUATE, 9-74
PICK/BASIC - EXECUTE, 9-75
PICK/BASIC - EXP, 9-77
PICK/BASIC - FIELD, 9-80
PICK/BASIC - FOOTING, 9-81
PICK/BASIC - FOR...NEXT, 9-83
PICK/BASIC - GOSUB, 9-87
PICK/BASIC - GOTO, 9-88
PICK/BASIC - HEADING, 9-89
PICK/BASIC - ICONV, 9-91
PICK/BASIC - IF (multi-line), 9-94
PICK/BASIC - IF (single-line), 9-92
PICK/BASIC - INCLUDE, 9-91
PICK/BASIC - INPUT, 9-93
PICK/BASIC - input functions, 9-95
PICK/BASIC - INT, 9-97
PICK/BASIC - LEN, 9-98
PICK/BASIC - LN, 9-99
PICK/BASIC - LOCK, 9-102
PICK/BASIC - LOOP, 9-104
PICK/BASIC - MAT, 9-106
PICK/BASIC - MATREAD, 9-108
PICK/BASIC - MATREADU, 9-109, 9-110
PICK/BASIC - MATWRITE, 9-111
PICK/BASIC - MATWRITEU, 9-112
PICK/BASIC - MOD, 9-138
PICK/BASIC - NULL, 9-114
PICK/BASIC - NUM, 9-115
PICK/BASIC - OCONV, 9-116
PICK/BASIC - ON...GOSUB, 9-87
PICK/BASIC - ON...GOTO, 9-117
PICK/BASIC - OPEN, 9-118
PICK/BASIC - PAGE, 9-119
PICK/BASIC - PRINT, 9-122
PICK/BASIC - PRINTER ON/OFF/CLOSE, 9-125
PICK/BASIC - PROCREAD, 9-126
PICK/BASIC - PROCWRITE, 9-127
PICK/BASIC - PROMPT, 9-128
PICK/BASIC - READ, 9-130
PICK/BASIC - READU, 9-133, 9-135
PICK/BASIC - READVU, 9-133, 9-135
PICK/BASIC - RELEASE, 9-137
PICK/BASIC - REM, 9-138
PICK/BASIC - RETURN {TO}, 9-140
PICK/BASIC - REWIND, 9-141
PICK/BASIC - RND, 9-142
PICK/BASIC - RQM, 9-147
PICK/BASIC - SEQ, 9-145
PICK/BASIC - SIN, 9-146
PICK/BASIC - SLEEP, 9-147
PICK/BASIC - SPACE, 9-148
PICK/BASIC - STOP, 9-150
PICK/BASIC - STR, 9-151
PICK/BASIC - SUBROUTINE, 9-47
PICK/BASIC - SYSTEM, 9-152
PICK/BASIC - TAN, 9-155
PICK/BASIC - TRIM, 9-157
PICK/BASIC - UNLOCK, 9-158
PICK/BASIC - WEOF, 9-159
PICK/BASIC - WRITE, 9-160

PICK/BASIC - WRITET, 9-161
PICK/BASIC - WRITEU, 9-162
PICK/BASIC - WRITEVU, 9-162
PICK/BASIC - XTD, 9-164
PICK/BASIC compiler, 9-15
PICK/BASIC compiler options, 9-15
PICK/BASIC EXECUTION from PROC, 9-22
PICK/BASIC program - Compiling, 9-15
'PN' option, 6-56
POINTER FILE, 2-31
POINTER-FILE, 10-12, 10-14
Positioning buffer pointers, 5-15
POVF, 10-34
POWER function, 9-129
PP, 5-39
Precedence of operators, 9-25
PRECISION declaration, 9-120
Predefinition example, 7-24
Predefinition of print files, 7-20
PRESTORE command (P), 4-37
Prestore command length, 4-37
Prestore facility, 4-37
Prestores, repeating, 4-38
Primary file space, 2-11, 2-13
PRINT command in RUNOFF, 8-13
Print file being output, 7-74
Print file control data in the PROC secondary input buffer, 7-28
Print file control record, 7-60, 7-62
Print file control record identifier, 7-62
Print file copy count specification, 7-15
Print file creating line number, 7-62
Print file creation account, 7-62
Print file creation date, 7-62
Print file creation time, 7-62
Print file definition, 7-15
Print file deletion, 7-40
Print file deletion: forcing, 7-40
Print file disenquement error messages, 7-59
Print file disenqueuement examples, 7-56
Print file enquement timing, 7-17
Print file generation account, 7-74
Print file inspection, 7-26, 7-31
Print file length, 7-74
Print file output priority, 7-45
Print file predefinition, 7-15, 7-20
Print file predefinition example, 7-24
Print file queue, 7-19
Print file scheduling, 7-45
Print file selection, 7-25
Print file selection: SP-EDIT, 7-25
Print file size, 7-62
Print file status, 7-62
Print file storage, 7-40
Print file storage release, 7-17
Print file termination error messages, 7-58
Print file termination examples, 7-54
Print file to data file, 7-39
Print file to data file conversion, 7-38
Print file to tape, 7-15, 7-25, 7-27, 7-35
Print file to tape , 7-37
Print file to tape example, 7-23
Print file to tape record length, 7-37
Print file to tape: messages, 7-37
Print file to user terminal, 7-38
Print file: Display, 7-36
Print file: enqueuing, 7-37
Print files: closing, 7-20
Print files: controling storage use, 7-17
Print files: disenqueuing, 7-52, 7-55
Print files: killing, 7-52
Print files: killing, 7-53
Print files: multiple, 7-20
Print files: termination, 7-53
PRINT INDEX in RUNOFF, 8-13
Print jobs: terminating, 7-52
Print limiters, 6-47
PRINT ON, 7-16
PRINT ON example, 7-24
PRINT statement, 9-122
PRINT-ERR, 10-16
PRINT-ON assignment specification, 7-15
PRINT-ON specification, 7-20
Printer, 7-62
Printer alignment, 7-41, 7-43
Printer allocation, 7-74
Printer characteristics, 3-25
Printer copy, 2-40
Printer detachment error messages, 7-59
Printer device, 7-68
Printer device address, 7-41
Printer initialization, 7-41
Printer line number, 7-68
Printer line specification, 7-41
PRINTER ON/OFF/CLOSE statements, 9-125
Printer ordinal, 7-41, 7-74
Printer output: suppression, 7-15
Printer pagination, 3-25
Printer reinitialization, 7-41
Printer status, 7-68, 7-74
Printer type, 7-41, 7-68, 7-74
Printers

Characteristics, 3-27
Configuration, 12-39
Printers: logical detachment, 7-52, 7-57
Printing multiple copies, 7-19
Priority: print file output, 7-45
Privilege level, 10-19
PROC and HOLD ENTRY #, 7-25
PROC and SP-EDIT, 7-25, 7-28
PROC command summary, 5-11
PROC control of SP-EDIT, 7-28
PROC control: STOPPTR, 7-49
PROC execution, 5-6
PROC i/o buffers, 5-8
PROC secondary input buffer, 7-28, 7-89
PROC secondary input buffer use, 7-88
PROCLIB file, 10-12
PROCREAD statement, 9-126
PROCWRITE statement, 9-127
Program, 9-12, 9-15
Programming examples, 9-176, 9-177, 9-178, 9-179, 9-181
> prompt, 3-3
PROMPT statement, 9-128
Propagate operator, F - code, 6-117
Purging a file, 2-32, 2-35
Purging an account, 10-36
Putting a terminal to sleep, 3-22
PW, 5-39
PX, 5-39
Q-items, 2-7, 2-20, 2-22, 2-28
Q-pointers, 2-22, 2-24
Creating with SET-FILE, 3-20
Q-pointers to multiple data files, 2-25
QFile, 3-20
QSELECT verb, 6-23, 6-88
R - code, 6-105
R - repeat operand, 6-120
R indicator, 7-62, 7-64
R option, 7-25, 7-28, 7-31, 7-37, 7-62
'R' option, 6-64
Random file access, 2-5
READ command in RUNOFF, 8-13
READ statement, 9-130
Reading a T-DUMP tape, 6-93
Reading a tape, 5-19
Reading the terminal, 5-19
READNEXT in RUNOFF, 8-13
READNEXT statement, 9-131
READU statement, 9-133, 9-135
READV statement, 9-136
READVU statement, 9-133, 9-135
Reallocation parameter, 2-20
REFORMAT - ACCESS file updating, 6-79
Reformat files - REFORMAT & SREFORMAT, 6-79
REFORMAT verb, 6-79
Relational connective -- default, 6-28
Relational connective: default, 6-33
Relational connectives, 6-27, 6-34
Relational expressions, 9-29, 9-31
Relational operators, 6-19, 9-29
Relational testing, 5-31
RELEASE statement, 9-137
REM function, 9-138
Remarks - PICK/BASIC, 9-12
Removing a printer, 7-52, 7-57
Removing strange item-ids, 6-16
REPEAT statement, 9-104
REPLACE COMMAND-EDITOR, 4-23
REPLACE function, 9-139
REPLACE UNIVERSAL - EDITOR, 4-23
Requeued, 7-62
Requeued print files, 7-64
Resetting buffers, 5-25
Restart option, 10-19
Restarting a printer, 7-41
Restoring an account, 10-51
Restoring PC Systems, 12-8
Restoring the system, 10-41, 10-42, 10-44
Retrieval locks, 10-19, 10-21
Return from subroutine, 5-39
RETURN {TO} statements, 9-140
REWIND statement, 9-141
Rewinding tape, 7-102
RI, 5-25
] - right-bracket, 6-26
Right-bracket (]), 6-26
Rn specification, 7-20, 7-21
RND function, 9-142
RO, 5-25
RQM statement, 9-147
Rules for generating ACCESS statements, 6-9
Run-time error messages, 9-192
Running a PICK/BASIC program, 9-20
RUNOFF - .* Command, 8-7
RUNOFF - comments, 8-7
RUNOFF - hyphens, 8-9
RUNOFF C option, 8-6, 8-13
RUNOFF Commands, 8-5
RUNOFF I option, 8-6, 8-13
RUNOFF Introduction, 8-3
RUNOFF J option, 8-9
RUNOFF options, 8-3
RUNOFF S option, 8-20
RUNOFF Tab Setting, 8-19
RUNOFF Underlining, 8-19
RUNOFF Upper/Lower Case, 8-19

Summation operator, F - code, 6-117
Summing an attribute value, 6-82
Suppress print output, 7-15, 7-16
Suppressing control-break data, 6-61
Suppressing detail with control-breaks, 6-66
SVM, 2-15
Synonym attribute names, 6-10
Synonym file definition items, 2-22
SYS-GEN tape format, 10-41
SYSPROG account, 10-34
System configuration, 10-53
System Date, 3-19, 12-20
System dictionary, 2-3
SYSTEM file, 10-12
System files, 2-30
SYSTEM function, 9-152
System Performance, 12-12, 12-39
System privileges, 10-19
System restore, 10-41, 10-42, 10-44
System software, 10-56
System status, 10-53
System Time, 3-21
SYSTEM-level files, 10-12
T - code, 6-109
T indicator, 7-62, 7-63
T option, 7-25, 7-27, 7-37
T option and PROC, 7-28
T response, 7-35, 7-38
T response during SPOOL T, 7-38
T specification, 7-16, 7-21, 7-37
'T' option, 6-56
T-ATT, 7-97, 7-99
T-BCK, 7-102
T-CHK, 7-104
T-conversion, 6-35
T-DET, 7-101
T-DUMP, 7-105
T-DUMP tape record size default, 7-97
T-DUMP verb, 6-93
T-EOD, 7-102
T-ERASE, 12-17
T-FWD, 7-102
T-LOAD, 7-105
T-LOAD verb, 6-93
T-RDLBL, 7-113
T-READ, 7-108, 7-109
T-RETEN, 12-17
T-REW, 7-102, 12-18
T-SPACE, 7-102
T-STATUS, 12-18
T-WEOF, 7-104
TA, 12-27
Tab Setting Characters in RUNOFF, 8-19
Table look-ups, 6-114
Table of Contents in RUNOFF, 8-7
TABS, 3-23, 8-22
Tabs in RUNOFF, 8-22
TABS Verb, 3-23
TAN function, 9-155
Tape, 7-62
Tape attachment, 7-97, 7-99
Tape control in PROC, 7-90
Tape detachment, 7-101
Tape dump to printer, 7-108, 7-109
Tape dump to terminal, 7-108, 7-109
Tape error message numbers, 7-90
Tape i/o verbs, 6-93
Tape input, 5-19
Tape label creation, 7-113
Tape label problem, 10-43
Tape label suppression, 7-27, 7-37
Tape label use, 7-113
Tape labels, 7-16, 7-113
Tape labels -- Hold file to tape, 7-27
Tape labels: supression, 7-25
TAPE modifier, 6-93
Tape output, 7-15, 7-16
Tape print file specification example, 7-23
Tape print files: retreival, 7-111
Tape record size, 7-16, 7-27, 7-97, 7-99
Tape record size defaults, 7-99
Tape record size for print files, 7-27
Tape record size with SP-EDIT, 7-27
Tape record size: hold files, 7-38
Tape verbs, 7-13
Tape: backspacing, 7-102
Tape: checking after write, 7-104
Tape: forward spacing, 7-102
Tape: general considerations, 7-93
Tape: reading files, 7-105
Tape: rewinding, 7-102
Tape: specifying block size, 7-97, 7-99
Tape: writing end-of-file, 7-104
Tape: writing files, 7-105
TB command - Editor, 4-31
TCL level, 3-3
TCL processors, 3-3
TCL prompt, 3-3
TCL statements, 3-3
TCL verb definitions, 3-12
TCL verbs, 3-3
  FORMAT, 12-17
  PASSWORD, 3-18
  SET-DATE, 3-19
  SET-DATE-STD, 12-20
  SET-FILE, 3-20
  SET-LPTR, 12-39

SP-EDIT to tape, 7-27
SP-EDIT to tape in PROC, 7-90
SP-EDIT: Copy count change, 7-28
SP-EDIT: Form number change, 7-28
SP-EDIT: form number selection, 7-30
SP-EDIT: output queue change, 7-28
SP-EDIT: output queue selection, 7-30
SP-EDIT: print file inspection, 7-31
SP-EDIT: tape record size, 7-27
SP-EDIT: without the prompts, 7-28
SP-EDITing all hold files, 7-30
SP-KILL, 7-52, 7-53
SP-KILL D, 7-52, 7-57
SP-KILL D error messages, 7-59
SP-KILL D examples, 7-57
SP-KILL error messages, 7-58, 7-59
SP-KILL examples, 7-54
SP-KILL F, 7-52, 7-55
SP-KILL F examples, 7-56
SP-KILL options, 7-53
SP-OPEN, 7-15, 7-19, 7-20
SP-STATUS verb, 7-74
SP-STOP replacement, 7-49
SP-TAPEOUT verb, 7-111
SPACE command in RUNOFF, 8-18
SPACE function, 9-148
SPACING command in RUNOFF, 8-18
Special <CTRL> characters, 3-4
Special characters, 8-20
Special Characters in RUNOFF, 8-19
SPOOL F, 7-39
SPOOL F and the form of the resulting item, 7-39
SPOOL F: item sequence, 7-39
SPOOL F: maximum item size, 7-39
SPOOL T, 7-38
SPOOL T control, 7-38
SPOOL T exit, 7-38
SPOOL T responses, 7-38
SPOOL TN, 7-38
SPOOL TN and form-feeds, 7-38
SPOOL TN and the TERM verb, 7-38
Spool to tape example, 7-23
SPOOL: SP-EDIT, 7-35, 7-37
Spooled, 7-62
Spooler activity, 7-74
Spooler entry number, 7-62
Spooler verbs, 7-13
Spooler: initialization, 7-81
Spooler: reinitialization, 7-81
Spooling -- forced, 7-28
Spooling print files to a printing terminal, 7-38
Spoolng a print file, 7-37
SQRT function, 9-149
SREFORMAT - ACCESS file updating, 6-79
SREFORMAT verb, 6-79
SS select secondary input buffer, 5-13
SSELECT verb, 6-23, 6-84
ST select output buffer, 5-13
Stacked Output in PROCs, 5-25
STANDARD command in RUNOFF, 8-18
Standard RUNOFF settings, 8-18
Start buffer mark, 4-37
Starting a printer, 7-41
STARTPTR and coldstarts, 7-43
STARTPTR error messages, 7-46
STARTPTR verb, 7-41
STARTSPOOLER, 7-81
:STARTSPOOLER verb, 7-81
STAT verb, 6-83
STAT-FILE, 10-37
Stat-file report, 10-47
Statement labels, 9-12
Statement numbers, 5-27, 5-29
Statistics of a file, 6-91, 6-92
Statistics of an attribute value, 6-83
Statistics of files, 10-37
Statistics of hashing, 10-30
Status indicators: print file, 7-63
Status of the system, 10-53
STOFF select primary output buffer, 5-13
STON select secondary output buffer, 5-13
Stop Bits, 12-26
STOP statement, 9-150
Stoping a printer, 7-49
STOPPED, 7-68
STOPPTR error messages, 7-51
STOPPTR verb, 7-49
STR function, 9-151
streaming tape, 12-16
String, 9-23
String expressions, 9-27
String functions, 6-117
String searching, 6-26
STRING-: SP-EDIT, 7-35
STRINGS - EDITOR, 4-9
Sub-lists using WITHIN, 6-75
Sub-strings, 9-27, 9-80, 9-92
Sub-strings, LOCATE, 9-100
Sub-values, 2-15
Subroutine, 5-43
SUBROUTINE statement, 9-47
Subroutines, 9-47, 9-87, 9-140
Subroutines - External, 9-47
SUBTOTALS - generating, 6-63
SUM verb, 6-82
Summary of dictionary items, 2-28

RUNOFF', 8-19
S, 5-15
S - code, 6-107
S command - Editor, 4-31
S indicator, 7-62, 7-63
S option, 6-96, 7-25, 7-28, 7-41
S option in BASIC, 9-165
S response, 7-35, 7-36
S specification, 7-16, 7-21
S-DUMP, 7-105
S? Command - Editor, 4-35
Sample PROC, 5-45, 5-46, 5-47
SAVE INDEX command in RUNOFF, 8-17
SAVE-LIST verb, 6-86
Saving an account, 10-51
Saving data, 10-49
SCT Support, 12-12
SECTION command in RUNOFF, 8-17
Security, 10-19, 10-21
Security codes, 10-21
SEL-RESTORE, 10-45
SELECT statement, 9-143
SELECT verb, 6-23, 6-84
Select-list, 3-11
Select-list testing, 5-35
Select-lists, 6-84, 6-86, 6-88
Selecting file extents, 2-13
Selecting items, 9-131, 9-143
Selecting MODULO, 2-13
Selecting output queues to SP-EDIT, 7-30
Selecting PROC buffers, 5-13
Selection - relational connectives, 6-34
Selection - value strings, 6-33
Selection -- data evaluation, 6-30
Selection by data value, 6-30
Selection criteria, 6-26, 6-42, 6-43
Selection criteria - formation, 6-24
Selection criterion, 6-30
SELECTION PROCESSOR, 6-27
Selective restore of files, 10-45
Semicolon - PICK/BASIC, 9-12
Sending a print file to a data file, 7-39
Sentences, 6-9
SEQ function, 9-145
Sequential file access, 2-5
Serial Port Characteristics, 12-28
Serial printer, 7-41, 7-68
Serial printer page length, 7-41
SET TABS in RUNOFF, 8-17
SET-DATE Verb, 3-19
SET-DATE-STD Verb, 12-20
SET-FILE Verb, 3-20
SET-FLOPPY, 12-16, 12-17, 11-8
SET-LPTR Verb, 12-39
SET-PORT Verb, 12-26
SET-SCT, 12-16, 11-8
SET-TIME Verb, 3-21
Setting buffer pointers, 5-15
SETTING TABS, 3-23
Sharing dictionaries, 2-9
Sharing PICK/BASIC programs, 9-21
Simple item-lists, 6-21
SINE function, 9-146
Single Character Output, 9-119
Single-level files, 2-3, 2-20, 2-33
SIZE field, 6-10
SIZE? - Editor, 4-35
SK command in RUNOFF, 8-18
SKIP command in RUNOFF, 8-18
SLEEP (TCL-I verb), 3-22
SLEEP statement, 9-147
Sort - Exploding, 6-72
Sort Keys, 6-69
Sort Specifications
  MS Processing Code, 6-131
SORT verb, 6-69
SORT-ITEM verbs, 6-96
SORT-LABEL verb, 6-76
Sorting by multi-values, 6-72
SP command in RUNOFF, 8-18
SP select primary input buffer, 5-13
SP-ASSIGN, 7-15
SP-ASSIGN -- General Form, 7-21
SP-ASSIGN ? assignment interrogation, 7-19
SP-ASSIGN and SP-EDIT, 7-37
SP-ASSIGN C example, 7-23
SP-ASSIGN changes, 7-15
SP-ASSIGN default, 7-22
SP-ASSIGN examples, 7-22
SP-ASSIGN Fn -- output queue, 7-19
SP-ASSIGN n -- copy count, 7-19
SP-ASSIGN Rn -- print file predefinition, 7-20
SP-ASSIGN Rn example, 7-24
SP-ASSIGNment display, 7-73
SP-ASSIGNment parameter change, 7-31
SP-CLOSE, 7-15, 7-19, 7-20
SP-EDIT, 7-25
SP-EDIT examples, 7-33
SP-EDIT exit, 7-36
SP-EDIT look, 7-26
SP-EDIT options, 7-25
SP-EDIT print file selection, 7-25
SP-EDIT prompts, 7-35
SP-EDIT R, 7-62
SP-EDIT termination messages, 7-26

SET-TIME, 3-21
TABS, 3-23
TERM-TYPE, 3-27
TCL-I VERBS, 3-10
TCL-II VERBS, 3-11
TCL-PROC interface, 5-6
TERM, 3-25
TERM-TYPE Verb, 3-27
Terminal Characteristics, 3-25
Terminal copy, 2-40
Terminal cursor control, 5-23, 9-38
Terminal Definitions, 12-38
Terminal emulating a serial printer, 7-38
Terminal input, 5-19, 9-93
Terminal output, 5-21
Terminal output, Formated, 5-23
Terminal type code, 3-25
Terminals
Characteristics, 3-27
Terminating a PROC, 5-39
Terminating execution, 3-31
Termination of PICK/BASIC program, 9-72
Test for existence, 6-31
TEST PAGE in RUNOFF, 8-18
Test-Cursor', 12-38
Testing item-ids, 6-27
Testing parameters, 5-31, 5-33
Text extraction, 6-109
Text strings, 5-25
Tfile - code, 6-114
THE - modifier, 6-54
The TERM verb and SP-EDIT, 7-38
Throwaway modifiers, 6-54
Time
Setting, 3-16, 3-21
TIME (TCL-I Verb), 3-28
Time conversion - input, 6-35
Time format, 6-113
TIME() function, 9-156
TIMEDATE() function, 9-156
TN response, 7-35, 7-38
TOP command - Editor, 4-12
TOTAL - evaluation sequence, 6-59
TOTAL - modifier, 6-58
Total limiters, 6-58
TOTAL modifier, 6-63, 6-64
Trace table - PICK/BASIC debugger, 9-169
Trailing blank lines -- truncation, 7-28
Transfer of control, 5-27
Translate conversion, 6-35
Translation of attribute data, 6-114
TRIM function, 9-157
TW option group and PROC, 7-28
Type-Ahead Capability, 12-27
U, 8-18
U - code, 6-116
U option, 7-25, 7-111
U option in SP-EDIT, 7-30
U response during SPOOL T, 7-38
'U' grand-total option, 6-60
'U' option, 6-64
UC command in RUNOFF, 8-18
UNALLOCATED, 7-68
Unconditional branch, 9-88
Underlining, 8-20
Underlining in RUNOFF, 8-19
UNLOCK statement, 9-158
Unlocked hold files, 7-26
Unprintable Characters, 4-36
UP command - Editor, 4-11
^ - up-arrow, 6-26
Up-arrow (^), 6-26
UP-ARROW - EDITOR, 4-10
Update locks, 10-19, 10-21
Updating single attributes, 9-136
Upgrading PC Systems, 12-8
Upper and lower case, 8-19
Upper case control, 8-19
UPPER CASE in RUNOFF, 8-18
Upper/Lower Case in RUNOFF, 8-19
USA Date Format, 12-20
User Accounts
Passwords, 3-18
USER conversions, 6-116
User identfication items, 10-12
User identification items, 10-19
User terminal as serial printer, 7-38
USING connective, 6-13
V option, 7-25, 7-28, 7-39
'V' option, 6-64
Value phrase, 6-33
Value phrases, 6-40
Value phrases: ANDed, 6-41
Value string, 6-30
Value strings, 6-40
Value taken from data, 6-31
Value-lists - Formation, 6-24
Values, 2-15
Values -- selection by, 6-30
Variable length records, 2-3
Variable values - Clearing, 9-53
Variables, 9-23
Vectors, 9-67
Verb definitions, 3-3
Verb formats, 3-12
Verbs, 7-13
VERIFY-SYSTEM, 10-56

Virtual memory, 10-3
VM, 2-15
W option, 7-25, 7-37, 7-49
WEOF statement, 9-159
WHAT, 10-53
WHERE, 10-53
WHO (TCL-I verb), 3-30
Wildcard Toggle - Editor, 4-36
WITH connective, 6-30
WITH modifier, 6-24
With phrase, 6-30
? with SP-ASSIGN, 7-19
WITHIN modifier, 6-75
WITHOUT connective, 6-30
WITHOUT EACH, 6-30
Work-space, additional, 10-6
WORKSPACE BEING LINKED; WAIT, 10-6
WRITE statement, 9-160
WRITET statement, 9-161
WRITEU statement, 9-162
WRITEV statement, 9-163
WRITEVU statment, 9-162
Writing the terminal, 5-21
X, 5-39
X indicator, 7-53, 7-62, 7-64
X option, 7-41
X response, 7-35
X response during SPOOL T, 7-38
XCS, 12-27
XTD function, 9-164
Y response, 7-35, 7-36, 7-37
Z -debug command, 9-173
Z command - Editor, 4-31

1691 Browning Irvine, CA 92714 • (714) 261-7425 Telex: 655420