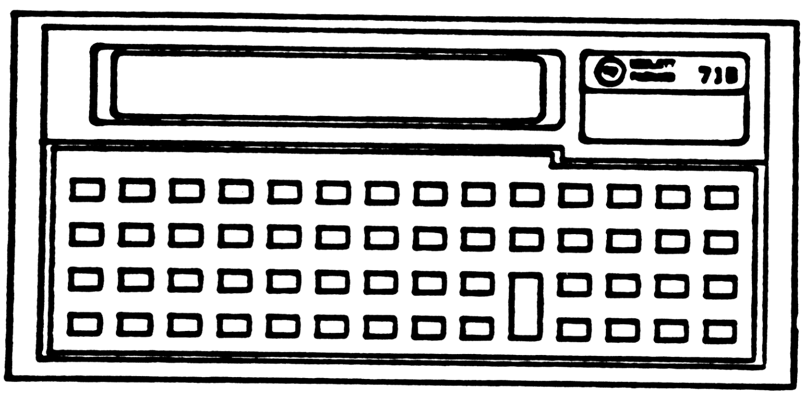
71-00015-8

# HEWLETT PACKARD USERS' LIBRARY

## LEX File Utilities

FOR THE HP-71B



# 71-00002 PROGRAM DESCRIPTION

Program Title ROWCOL

Contributor Hewlett-Packard Company

Address 1000 NE Circle Blvd

City Corvallis State Oregon Country U.S.A.

Telephone Zip/Postal Code 97330

Program Description (include equations) This lex file provides one keyword: ROWCOL$. Invocation:

ROWCOL$(<graphstring>)

The keyword accepts a single string argument of 0-8 characters. If argument is n characters (n<8) then characters n+1 through 8 default to nulls. Argument of >8 characters causes an "Invalid Arg" error.

Argument represents an 8 pixel by 8 pixel block of row- or column-oriented graphics. Result is an 8 pixel by 8 pixel block of column- or row-oriented graphics, respectively.

An argument or result of row-oriented graphics would actually be 8 bytes each containing 8 bits of column data from consecutive rows.

An argument or result of column-oriented graphics is actually 8 bytes each containing 8 bits of row data from consecutive columns. Here is a more hands-on explanation:

Necessary Accessories None

Supported Accessories N/A

Operating limits and warnings

File name(s) ROWCOL

Size of file(s) 119 bytes Additional RAM Requirement to run the program None

References

+------------------------------------------------------------------------+

# CHAPTER 3
# ROWCOL

+------------------------------------------------------------------------+

New Lex File Indicates "RC:A" in VER$ string.

Program Title: String function for row/column graphics conversion.

Category Number(s): ???

File Name(s): ROWCOL.

Primary Category Name: ???

Size of File(s): 119 bytes.

Additional RAM Requirement: None.

Abstract: Lex file provides a keyword that allows easy conversion between row- and column-oriented graphics. Sample use: converting graphics data for HP82905B printer (column-oriented) into graphics data for Thinkjet printer (row-oriented).

Necessary Accessories: None.

Supported Accessories: N/A.

## 3.1 Program Description

This lex file provides one keyword: ROWCOL$. Invocation:

ROWCOL$(<graphstring>)

The keyword accepts a single string argument of 0-8 characters. If argument is n characters (n<8) then characters n+1 through 8 default to nulls. Argument of >8 characters causes an "Invalid Arg" error.

Argument represents an 8 pixel by 8 pixel block of row- or column-oriented graphics. Result is an 8 pixel by 8 pixel block of column- or row-oriented graphics, respectively.

An argument or result of row-oriented graphics would actually be 8 bytes each containing 8 bits of column data from consecutive rows.

### 3.2 Variable Definitions

N/A. 71-00002

### 3.3 Sample Usage

The following program converts a textfile containing graphics information for a THINKJET printer into graphics information for an HP82905B printer and prints that information.

The program is not fast; each line of print on the HP82905B takes about 45 seconds. But the use of the ROWCOL$ function on line 280 produces a drastic speed increase over what the program would take if it performed the equivalent manipulations in BASIC.

The program assumes that the file being dumped (called "MYFILE" here) contains THINKJET graphics directives of the form "<esc>*b<#bytes>W" (the preamble) followed by bytes of row graphics information. Any lines not of this format (as typically occur at the beginning and end of such files) are discarded without resulting in anything being printed.

# 71-00002 SAMPLE PROBLEM SOLUTION

| DISPLAY CONTENTS | USER RESPONSE | COMMENTS |
| --- | --- | --- |

```
10 PRINT CHR$(27)&"&k2S"&CHR$(27)&"&l9d0L"
20 PWIDTH INF
30 DESTROY ALL
40 OPTION BASE 1
50 DIM C$[800],L(8),T$[8]
60 ASSIGN #1 TO MYFILE
70 ON ERROR GOTO 320
80 DESTROY R$ @ DIM R$(8)[100]
90 FOR I=1 TO 8
100 READ #1;R$(I)
110 IF R$(I)[1,3]#CHR$(27)&"*b" THEN 100
120 R$(I)=R$(I)[POS(R$(I),"W")+1]
130 NEXT I
140 OFF ERROR
150 GOSUB 170
160 GOTO 70
170 L9=0
180 FOR I=1 TO 8
190 L(I)=LEN(R$(I))
200 L9=MAX(L9,L(I))
210 NEXT I
220 C$=""
230 FOR I=1 TO L9
240 T$=""
250 FOR J=1 TO 8
260 T$=T$&R$(J)[I,I]&CHR$(0)[1+(I<=L(J))]
270 NEXT J
280 C$=C$&ROWCOL$(T$)
290 NEXT I
300 PRINT CHR$(27)&"*b"&STR$(LEN(C$))&"G";C$
310 RETURN
320 OFF ERROR
330 GOSUB 170
340 END
```

# 71-00002 SAMPLE PROBLEM

The followin program converts a textfile containing graphics information for a THINKJET printer into graphics information for an HP 82905B printer and prints that information.

The program is not fast; each line of print on the HP 82905B takes about 45 seconds. But the use of the ROWCOL$ function on line 280 produces a drastic speed increase over what the program would take if it performed the equivalent manipulations in BASIC.

The program assumes that the file being dumped (called "MYFILE" here) contains THINKJET graphics directives of the form "<esc>*b<#bytes>W" (the preamble) followed by bytes of row graphics information. Any lines not of this format (as typically occur at the beginning and end of such files) are discarded without resulting in anything being printed.

Line 10 initializes the HP 82905B printer to compressed print mode, 9 lpi spacing and no-perforation-skip; appropriate settings for many graphics dumps. Lines 20-60 initialize the program. Line 70 traps the end-of-file condition. Line 80 reinitializes the row graphics string array to nulls. Lines 90-130 accumulate 8 rows of row graphics information for conversion to column graphics. Line 150 calls the subroutine to perform the actual conversion and line 160 loops back for more graphics information.

Lines 320-340 handle the printing of the final rows when the end-of-file is reached.

The conversion work is done in the subroutine at lines 170-310. Lines 170-210 build an array containing the lengths of the graphics data in each row, with L9 = the maximum length (note that line 120 stripped off the row graphics preamble from the line). Line 220 initializes the column graphics string to empty. Lines 230-290 build the column graphics string by grouping the row graphics bytes properly into T$ (line 260 insures a null space-filler if a row string is too short), converting T$ into column form with ROWCOL$ and appending it to the column graphics string (line 280). Line 300 prints the column graphics information prepended with the proper column graphics preamble ("<esc>*b<#bytes>G").

With slight modification, this program could print to a file instead of to a printer, producing a text file that can be easily and quickly printed on a column graphics printer.

# 71-00002

(Continuation Page)

Consider This pixel pattern:

```
                                              Row graphics pattern:
                    -------------------
                    | x   x x     x   |  4D
   (all numbers     |   x x x   x x x |  EE
     in HEX)        | x x   x x     x |  9B
                    | x x x   x x x   |  77
                    |     x x x x x x |  FC
                    | x   x x x x   x |  BD
                    | x   x   x x x x |  F5
                    | x x x         x |  87
                    -------------------
   Column     :       E 8 F 3 7 7 5 F
   graphics   :       D E B 7 C A B 6
   pattern    :
```

The ROWCOL$ function will convert the byte sequence ED8EFB377C7A5BF6h into 4DEE9B77FCBDF587h and will also do the inverse (it is its own inverse).

This function provides a tool for BASIC to speak both row- and column-oriented graphics with minimal headache.

# 71-00002

Line 10 initializes the 82905B printer to compressed print mode, 9 lpi spacing and no-perforation-skip; appropriate settings for many graphics dumps. Lines 20-60 initialize the program. Line 70 traps the end-of-file condition. Line 80 reinitializes the row graphics string array to nulls. Lines 90-130 accumulate 8 rows of row graphics information for conversion to column graphics. Line 150 calls the subroutine to perform the actual conversion and line 160 loops back for more graphics information.

Lines 320-340 handle the printing of the final rows when the end-of-file is reached.

The conversion work is done in the subroutine at lines 170-310. Lines 170-210 build an array containing the lengths of the graphics data in each row, with L9 = the maximum length (note that line 120 stripped off the row graphics preamble from the line). Line 220 initializes the column graphics string to empty. Lines 230-290 build the column graphics string by grouping the row graphics bytes properly into T$ (line 260 insures a null space-filler if a row string is too short), converting T$ into column form with ROWCOL$ and appending it to the column graphics string (line 280). Line 300 prints the column graphics information prepended with the proper column graphics preamble ("<esc>*b<#bytes>G").

With slight modification, this program could print to a file instead of to a printer, producing a text file that can be easily and quickly printed on a column graphics printer.

# 71-00003 PROGRAM DESCRIPTION

Program Title LIFE:A

Contributor Hewlett-Packard Company

Address 1000 NE Circle Blvd

City Corvallis State Oregon Country U.S.A.

Telephone Zip/Postal Code 97330

Program Description (include equations) A familiarity with the game LIFE is assumed.

This lex file contains one keyword: LIFE$. The keyword is a string function that takes a string argument representing a current generation of LIFE (on a rectangular board of arbitrary dimension) and computes the next generation. The keyword is invoked as follows:

LIFE$ (<boardstring>,<row width<,>wrap flag>[,<alt fill char>])

The input parameters are as follows:

<boardstring> String parameter representing the current board. If the board is, for example, 80 columns by 24 rows, the string is 1920 characters long. The cells are in row-major order; that is, assuming the 80 x 24 case, the first 80 characters represent the first row, the next 80 characters represent the second row, etc. Empty cells are represented by a blank, occupied cells are represented by the fill character (which defaults to an "*" if not specified in the fourth parameter).

Necessary Accessories None

Supported Accessories N/A

Operating limits and warnings

File name(s)

Size of file(s) Additional RAM Requirement to run the program

References

# CHAPTER 1
# LIFELEX

New LEX File Indicates "LIFE:A" in VER$ string.

Program Title: LIFE Generation Computation Utility

Category Number(s): F102.

File Name(s): LIFELEX.

Primary Category Name: GAMES.

Size of File(s): 457 bytes.

Additional RAM Requirement: None.

Abstract: This lex file contains a very fast next-generation computer for John Conway's "Life" game. Computation time for a 24 by 80 board (useful if output is going to a typical terminal) is typically under two seconds.

Necessary Accessories: None.

Supported Accessories: N/A.

## 1.1 Program Description:

A familiarity with the game LIFE is assumed.

This lex file contains one keyword: LIFE$. The keyword is a string function that takes a string argument representing a current generation of LIFE (on a rectangular board of arbitrary dimension) and computes the next generation. The keyword is invoked as follows:

```
LIFE$(<boardstring>,<row width>,<wrap flag>[,<alt fill char>])
```

The input parameters are as follows:

<boardstring> String parameter representing the current board. If the board is, for example, 80 columns by 24 rows, the string is 1920 characters long. The cells are in row-major order; that is, assuming the 80 x 24 case, the first 80 characters represent the first row, the next 80 characters represent the

second row, etc. Empty cells are represented by a blank, occupied cells are represented by the fill character (which defaults to an "*" if not specified in the fourth parameter).

<row width> Numeric parameter identifying row width. In the 80 x 24 case discussed above, this would be 80.

<wrap flag> Numeric parameter. Non-zero if the game board wraps around the edges. In other words, if non-zero, the top edge of the board is considered adjacent to the bottom edge and the left edge is adjacent to the right edge. If zero, the board edges are considered the "edge of the world".

<alt fill char> Optional string parameter. Specifies a character to use instead of "*" as the fill character.

The result of LIFE$ is a string representing the next generation playing board. Empty cells are represented by blanks, occupied cells by the fill character.

Error Conditions: LIFE$ will fail with an "Invalid Arg" error if any of the following is true:

- Either numeric argument is not a real finite scalar.
- <row width> is less than 3 or greater than 1048575.
- Length of <boardstring> is not an integer multiple of <row width>.
- Number of rows (length of <boardstring> divided by <row width>) is less than 3.

## 1.2 Variable Definitions

N/A.

## 1.3 Sample Usage

The following program demonstrates the use of LIFE$. It was written to send its output to an HP-82163A video interface, and requires an HPIL interface and an 82163A. The program uses cursor control sequences particular to that interface and makes assumptions about the screen size.

Lines 10-70 initialize variables. Lines 90-120 initialize the board to a random pattern (50% filled) of empty and occupied cells. Lines 130-140 print the current generation. Line 150 computes the next generation. Line 160 checks if the board has reached a one- or two-generation stability. The program terminates when one- or two-

# 71-00003 SAMPLE PROBLEM

The following program demonstrates the use of LIFE$. It was written to send its output to an HP 82163A video interface, and requires an HPIL interface and an HP 82163A. The program uses cursor control sequences particular to that interface and makes assumptions about the screen size.

Lines 10-70 initialize variables. Lines 90-120 initialize the board to a random pattern (50% filled) of empty and occupied cells. Lines 130-140 print the current generation. Line 150 computes the next generation. Line 160 checks if the board has reached a one- or two- generation stability. The program terminates when one- or two-generation stability is reached (on occasion, this may never occur).

```
10 DESTROY ALL @ RANDOMIZE @ PRINTER IS :DISPLAY
20 PWIDTH INF
30 OPTION BASE 1
40 DIM B$[480],X$(2)[480]
50 G=0
60 X$(1)="" @ X$(2)=""
70 B$=""
80 PRINT CHR$(27)&"H"&CHR$(27)&"JCONSTRUCTING BOARD..."
90 FOR I=1 TO 480
100 B$=B$&"* "[RND+1][1,1]
110 PRINT B$[I,I];
120 NEXT I
130 PRINT CHR$(27)&"H";B$;
140 PRINT "      GENERATION #";G;CHR$(27)&"J"; @ G=G+1
150 B$=LIFE$(B$,32,1)
160 IF B$#X$(1) AND B$#X$(2) THEN X$(2)=X$(1) @ X$(1)=B$ @ GOTO 130
170 PRINT CHR$(27)&"%"&CHR$(0)&CHR$(15)&"STABLE AT GENERATION #";G-1
180 END
```

71-00003

(Continuation Page)

<row width> Numeric parameter identifying row width. In the 80 x 24 case discussed above, this would be 80.

<wrap flag> Numeric parameter. Non-zero if the game board wraps around the edges. In other words, if non-zero, the top edge of the board is considered adjacent to the bottom edge and the left edge is adjacent to the right edge. If zero, the board edges are considered the "edge of the world".

<alt fill char> Optional string parameter. Specifies a character to use instead of "*" as the fill character.

The result of LIFE$ is a string representing the next generation playing board. Empty cells are represented by blanks, occupied cells by the fill character.

Error Conditions: LIFE$ will fail with an "Invalid Arg" error if any of the following is true:

Either numeric argument is not a real finit scalar.

<row width> is less than 3 or greater than 1048575.

Length of <boardstring> is not an integer multiple of <row width>.

Number of rows (length of <boardstring> divided by <row width>) is less than 3.

# 71-00004 PROGRAM DESCRIPTION

Program Title: Banner

Contributor: Hewlett-Packard Company

Address: 1000 NE Circle Blvd

City: Corvallis State: Oregon Country: U.S.A.

Telephone: Zip/Postal Code: 97330

Program Description (include equations): The lex file contains one keyword: BANNER$. BANNER$ is a banner-building tool. It takes a string argument of from 1 to 3 characters, and returns a 48-character string representing a 6x8 "banner" of the characters. For example, BANNER$ ("A") returns the string:

```
" AAA  A   A A   A AAAAA A   A A   A A   A       ",
```

which, when printed in 8 rows of 6 characters, is:

```
+------+
| AAA  |
|A   A |
|A   A |
|AAAAA |
|A   A |
|A   A |
|A   A |
|      |
+------+
```

(border shown for emphasis only)

Necessary Accessories: None

Supported Accessories: N/A

Operating limits and warnings:

File name(s): BANNER

Size of file(s): 202 bytes Additional RAM Requirement to run the program: None

References:

71-00004

2 of 4

+----------------------------------------------------------------------+

# CHAPTER 2
# BANNER

+----------------------------------------------------------------------+

New LEX File Indicates "BNR:A" in VER$ string.

Program Title: Banner-building utility.

Category Number(s): ???

File Name(s): BANNER.

Primary Category Name: ???

Size of File(s): 202 bytes.

Additional RAM Requirement: None.

Abstract: String keyword to create banner-type representations of characters in the built-in or alternate character sets. This allows easy printing of banners (posters) using large characters.

Necessary Accessories: None.

Supported Accessories: N/A.

## 2.1 Program Description

The lex file contains one keyword: BANNER$. BANNER$ is a banner-building tool. It takes a string argument of from 1 to 3 characters, and returns a 48-character string representing a 6x8 "banner" of the characters. For example, BANNER$("A") returns the string:

```
" AAA  A   A A   A AAAAA A   A A   A A       ",
```

which, when printed in 8 rows of 6 characters, is:

The following program implements a large-display clock on the screen of an HP 82163A video interface. It requires an HPIL interface and an HP 82163A.

The program works by figuring out the current time, converting to 12-hour format and "painting" the banner representations of the numbers and the am/pm indicators on the screen using cursor control escape sequences for the 82163A. The banner is built out of the CHR$ (160) character, which displays as a white block on the HP 82163A.

```
10 PRINTER IS :DISPLAY @ DISPLAY IS *
20 PWIDTH INF @ DELAY 0 @ DESTROY ALL
30 CLEAR :DISPLAY
40 T$="     "
50 M$="  "
60 U$=TIME$ @ T=VAL(U$[1,2])
70 IF T<12 THEN N$="am" ELSE N$="pm"
80 T=MOD(T-1,12)+1
90 U$[1,1]=" "
100 IF T<10 THEN U$[2,2]=STR$(T) ELSE U$[1,2]=STR$(T)
110 DISP U$[1,5]&" "&N$
120 FOR I=1 TO 5
130 IF T$[I,I]#U$[I,I] THEN CALL DSPDGT(U$[I,I],I,0)
140 NEXT I
150 T$=U$
160 FOR I=1 TO 2
170 IF M$[I,I]#N$[I,I] THEN CALL DSPDGT(N$[I,I],I+1.5,7)
180 NEXT I
190 M$=N$
200 WAIT 60-MOD(TIME,60) @ GOTO 60
210 SUB DSPDGT(D$,P,S)
220 DIM Z9$[48]
230 Z9$=BANNER$(D$[1,1]&CHR$(160))
240 FOR Z9=1 TO 8
250 PRINT CHR$(27)&"%"&CHR$(6*(P-1))&CHR$(Z9+S-1);
260 PRINT Z9$[Z9*6-5,Z9*6-1]&CHR$(27)&"<"
270 NEXT Z9
280 END SUB
```

71-00004

(Continuation Page)

If the argument is two characters long, the second character is used as an alternate building character, so BANNER$("A*") is:

```
+------+
| ***  |
|*   * |
|*   * |
|***** |
|*   * |
|*   * |
|*   * |
|      |
+------+
```

If the argument is three characters long, the third character is used as an alternate space character, so BANNER$("A*.") is:

```
+------+
|.***..|
|*...*.|
|*...*.|
|*****.|
|*...*.|
|*...*.|
|*...*.|
|......|
+------+
```

BANNER$ works for the built-in character set and for any characters defined in the alternate character set.

# 71-00005 PROGRAM DESCRIPTION

Program Title: Running Clock Display

Contributor: Hewlett-Packard Company

Address: 1000 NE Circle Blvd

City: Corvallis State: Oregon Country: U.S.A.

Telephone: Zip/Postal Code: 97330

Program Description (include equations): This lex file provides a running hh:mm:ss clock display that can be turned on or off. The clock occupies the 9 rightmost display positions and does not interfere with normal operation of the computer. That is, the computer can be used normally for running, editing, and so on while the clock is running.

The clock is invoked with the keyword sequence:

CLOCK ON

and is turned off with the keyword sequence:

CLOCK OFF

Some things to keep in mind about the clock:

CLKDISP

Necessary Accessories: None

Supported Accessories: N/A

Operating limits and warnings:

File name(s): CLKDISP

Size of file(s): 328 bytes Additional RAM Requirement to run the program: None

References:

71-00005



The clock performs an implicit "WINDOW 1,13" every time it ticks. It is therefore impossible to use the WINDOW command effectively while the clock is on.

The clock performs an implicit "WINDOW" 1,22" when CLOCK OFF is performed.

The clock is 24-hour format only.

During any operation requiring full CPU attention (such as performing a BEEP), the clock will stop running. The clock will NOT, however, lose time.

The clock display turns off when the calculator is turned off.

Sample Usage

```
10 CLOCK ON
20 CLOCK OFF
```

+----------------------------------------------------------------------+

CHAPTER 4
CLKDISP

+----------------------------------------------------------------------+

New Program Indicates "CLK:A" in VER$ string.

Program Title: Running clock display.

Category Number(s): ???

File Name(s): CLKDISP.

Primary Category Name: ???

Size of File(s): 328 bytes.

Additional RAM Requirement: None.

Abstract: Lex file provides an optional running clock display in the right-hand part of display. Clock does not interfere with normal operation of the computer.

Necessary Accessories: None.

Supported Accessories: N/A.

## 4.1 Program Description

This lex file provides a running hh:mm:ss clock display that can be turned on or off. The clock occupies the 9 rightmost display positions and does not interfere with normal operation of the computer. That is, the computer can be used normally for running, editing, and so on while the clock is running.

The clock is invoked with the keyword sequence:

CLOCK ON

and is turned off with the keyword sequence:

CLOCK OFF

Some things to keep in mind about the clock:

- The clock performs an implicit "WINDOW 1,13" every time it ticks. It is therefore impossible to use the WINDOW command effectively while the clock is on.
- The clock performs an implicit "WINDOW 1,22" when CLOCK OFF is performed.
- The clock is 24-hour format only.
- During any operation requiring full CPU attention (such as performing a BEEP), the clock will stop running. The clock will NOT, however, lose time.
- The clock display turns off when the calculator is turned off.

### 4.2 Variable Definitions

N/A.

### 4.3 Sample Usage

```
10 CLOCK ON
20 CLOCK OFF
```

# 71-00006 PROGRAM DESCRIPTION

Program Title Text File Utilities (TEXTUTIL)

Contributor HEWLETT PACKARD COMPANY

Address 1000 N.E. Circle Blvd.

City Corvallis State Or. Country U.S.A.

Telephone (503)757-2000 Zip/Postal Code 97330

Program Description (include equations) TEXTUTIL contains five new keywords, and an extension to the mainframe LIST. The five new keywords are: FILESZR - a function that returns the number of records in the specified text file. SEARCH - a function that searches through a TEXT file for the specified string, returning informatio as to if and where the string was found. DELETE# - a statement that allows a TEXT file record to be deleted. INSERT# - a statement that allows a record to be inserted into a TEXT file . REPLACE# - a statement that allows a TEXT file record to be replaced by another.

LIST is extended to list TEXT files.

Necessary Accessories None

Operating limits and warnings

Minimum RAM Requirement 1512 bytes

References

The following is a list of the keywords, their syntax and an example of the use of each. In most cases, the parameters are numeric expressions, string expressions or literals (refer to section 3 of the HP-71 Owner's Manual).

DELETE# channel number, record number

The DELETE# statement deletes the specified record from the text file associated with the channel number. Channel numbers are assigned with the ASSIGN# statement (refer to section 14 of the HP-71 Owner's Manual). Record numbers always begin at 0, so line number 1 is record number 0.

The channel number and the record number are numeric expressions, rounded to integer values.

DELETE# generates an error message if the assigned file is external, protected or not a text file.

EXAMPLE: DELETE# 11,14 deletes record number 14 from the file associated with channel 11.

FILESZR (filename)

The FILESZR function returns the number of records in the specified text file if that file exists. The filename parameter is a string expression. If an error is detected, the negated error number is returned so that you can tell the difference between an error and the number of records. If filename contains an illegal port specifier, such as FROGS:PORT(8), the error message Invalid Filespec is generated.

EXAMPLE:FILESZR ('FROGS') returns the number of records in the file FROGS.

INSERT# channel number, record number;new record

The INSERT# statement inserts the new record immediately before the specified record number in the file associated with the specified channel number. The channel number must first be assigned to the file using the ASSIGN# statement. Record numbers always begin at 0, so line 1 is record0.

The new record must be a string expression. The channel number and the record number are numeric expressions, rounded to integer values.

INSERT# generates an error if the file is external, protected or not a text file.

EXAMPLE: INSERT# 11,35;"This is the new line being inserted." inserts the string before record 35 (line 36) of the file associated with channel 11. The old record 35 becomes record 36.

LIST filename [begin line[end line]]

The LIST statement lists a text file. Depending on the parameters you specify, it lists either the entire file, a single line, or a range of lines. Line numbers are specified using integer constants. The line number parameters are optional, and the whole file is listed if they are not included. Refer to LIST in the HP-71 Reference Manual for details.

REPLACE# channel number, record number;new record

The REPLACE# statement replaces the record indicated by record number with the new record. The channel number must first be assigned to the file by using the ASSIGN# statement. Record numbers always begin at 0, so line number 1 is record 0.

The new record is a string expression. The channel number and the record number are numeric expressions rounded to integer values.

REPLACE# returns an error if the file is external, protected or not a text file.

EXAMPLE: REPLACE# 11,35;"This line will replace the old line."replaces record 35 of the file associated with channel 11. Old record 35 no longer exists.

SEARCH (search string, column, begin record, end record, channel number)

The SEARCH function searches the file associated with the indicated channel number for the search string, beginning with the specified column and record number. The search continues through the end record specified. If the search is successful, SEARCH returns a value in the form nnn.ccclll, where nnn is the record number, ccc is the column is the column number and lll is the length of matched string. If the search is unsuccessful, SEARCH returns a zero.

The search string can be any string expression, and can contain the special pattern characters discussed on the next page. The other parameters are numer expressions rounded to integer values.

EXAMPLE: Suppose that channel 11 has been assigned to the file FROGS and the string'frogs are green'appears beginning in column 8 of line 36.

A=SEARCH("frogs are green",1,1,9999,11)

searches the file FROGS, beginning with column 1 of rec. 1 through rec. 9999, for the search string and returns the value 35.008015 in A.

**Note that since the first line is record 0, line 36 is actually record 35.

Cont. next page

71-00006

SPECIAL PATTERNS
----------------

A feature of SEARCH is the availability of four characters that have special meaning when used in patterns. Using these characters in a search string tells SEARCH to look, for example, only for those occurrences of the string at the beginning of the line, or at the end of the line, or allow any pattern between two specified patterns. The four characters that can be used in this special way are ., @, ^, $.

The backslash (\) character can be used like a "switch" in the search string to start and stop this feature that makes these four characters take on special meaning. The backslash character is CHR$(92), and for convenience, may be assigned to a key by executing:

DEF KEY <key name>,CHR$(92);

(See page 69 in the HP-71 Reference Manual for further information about key assignments). The first occurrence of the backslash turns on the feature, so that the four characters take on their special meanings. The next occurence of the backslash turns this feature off.

The four characters, their meanings, and some examples of their use are described in the following paragraphs. In all the examples, assume that the specified file is open to channel number 3. Also, all the examples specify the search to start in record zero, column 1 (the start of the file), and to continue through record number 9999.

1) The period (.) is a 'wild card' character. SEARCH looks for the specified string, but any character can be in those positions in the string where you put a period.

   Example: SEARCH("ABC"&CHR$(92)&"...W",1,0,9999,3)
   Looks for the first occurrence of ABC followed by any three characters, followed by W. Possibilities are ABC999W, ABCzyzW, or ABC yzW

2) The commercial "at" symbol (@) indicates that any number of characters between the beginning of a string and the end of a string on the same line are 'wild cards' -- that is, there can be any number of characters-- you don't have to specify how many characters or what they are. Because SEARCH starts looking for the end of the string at the end of the line, the longest match is found.

   Example: SEARCH("ABC"&CHR$(92)&"@CDE",1,0,9999,3)
   Looks for the first occurrence of a string beginning with ABC and ending with CDE on the same line, such as ABC123CDE, ABCCDE, or ABC12 zzzCDE

3) The up-arrow (^) is used to find a string only when it occurs at the beginning of a line. If the string appears anywhere else in the line, it will be ignored. The up-arrow has this special meaning only when it appears as the first character of the string. Anywhere else in the string, ^ will have its normal meaning.

   Example: SEARCH("\^ABC",1,0,9999,3)
   Looks for the first occurrence of ABC only at the beginning of each line. If ABC appears anywhere else in the line, a match will not be found.

71-00006



4) The dollar sign ($) following the string causes SEARCH to look for the string only at the end of a line. The dollar sign character must appear at the end of the string. When it appears anywhere else in the string, it has its normal meaning.

71-00006

```
Example: SEARCH("ABC\$",1,0,9999,3)
         Looks for the first occurrence of ABC at the end of a line.
         If ABC appears anywhere else in the line, it will be ignored.
```

Sometimes, your string may contain a backslash character as part of the actual text. In this case, you don't want SEARCH to see the backslash as a switch. The solution is to use two sequential backslashes. SEARCH interprets \\ as a single backslash character, not as a switch.

# 71-00007 PROGRAM DESCRIPTION

Program Title Character Set LEX File Generator

Contributor Bruce Stephens

Address Hewlett-Packard Company (PCD), 1000 NE Circle Blvd.

City Corvallis State Oregon Country U.S.A.

Telephone 757-2000 Zip/Postal Code 97330

Program Description (include equations) This program creates a LEX file that contains an alternate character set of your design, and adds a keyword to activate that character set.

Necessary Accessories None

Supported Accessories N/A

Operating limits and warnings

File name(s)

Size of file(s) Additional RAM Requirement to run the program

References

# 71-00007 VARIABLE DEFINITIONS

| NAME | DEFINITION |
|---|---|
| F$ | Name of LEX file to be created (1 to 8 characters) |
| V$ | VER$ string of new LEX file (1 to 7 characters) |
| W$ | Name of new keyword (2 to 8 characters) |
| L1 | LEX id # of new LEX file |
| L2 | Token # of new keyword |
| L3 | Character set id # |
| C | Length of character set (in bytes) (6 bytes per character) |
| T | Length of LEX file (in bytes) |
| P$ | Holds string of hex digits to be put into new LEX file |
| FNH$ | Returns a character representing the hex value of argument |
| FNS$ | Returns a hex string, 2 digits for each character in argument. The least signficant nibble of the first byte of the argument occurs first, followed by the most significant nibble of the first byte. Successive bytes are appended after the first byte. |
| FNT$ | Same as FNS$ except successive bytes are inserted in front of preceding bytes, thus reversing the order of the bytes. |
| Z$ | Used by FNS$ and FNT$ to hold value to be returned. |

# 71-00007 SAMPLE PROBLEM

Create a character set by following the example in the HP-71 Owner's Manual on pages 133-135 (or create a character set of your own). When you are satisfied that you have the alternate character set as you like it (any number from 0 to 128 characters may be defined) run the example as shown on the following page.

When the example run has been completed the program will have created a new LEX file called TESTCH. To tell the system to look for the new LEX file, turn the machine off then back on.

Now display the VER$ function and you should see the string TST embedded somewhere in the string:

```
>VER$
HP71:1BBBB TST
```

To cancel the current alternate character set definition, type:

```
>CHARSET ""
```

The LEX file has added a new work (TESTCH) to the language. This keyword may be entered into a BASIC program or executed directly from the keyboard. To activate the character set, type:

```
>TESTCH
```

Now the character set is active. To display the first character in the special set type:

```
>CHR$(128)
```

The character set will remain active until the character set is redefined by another character set defining word is executed, the CHARSET statement is executed or the LEX file (TESTCHAR) is purged from system memory.

When the character set is activated, only 7 bytes of RAM is used in addition to the memory required to hold the LEX file. If the LEX file is in a ROM then only 7 bytes total is required to activate the character set.

# 71-00007 SAMPLE PROBLEM SOLUTION

| DISPLAY CONTENTS | USER RESPONSE | COMMENTS |
|---|---|---|
| > | RUN CHARSET | |
| New LEX file name: | TESTCHAR | Any valid file name |
| VER$ string: | TST | 1 to 7 characters |
| LEX id #(decimal): | 92 | 1 to 255 |
| Token #(decimal): | 5 | 1 to 255 |
| Charset id#(decimal): | 92 | 0 to 255 |

(Continuation Page)

This program prepares a string of hex digits which it POKE's into a file. This string must be exact to prevent locking up the machine or causing a memory lost condition. For this reason, the user should not attempt to modify this program unless he/she is quite familiar with the internals of the machine and understands the cryptic detail of the program.

The user should be careful not to select a LEX id/token # that conflicts with some other application that he is likely to run. LEX id numbers in the range 92-94 have been set aside for just such use by users creating their own LEX files. The user should be aware of possible conflicts with any other locally written lex files. If the token # is also defined by another LEX file with the same LEX id, the results are unpredictable and certainly undesirable.

Hewlett-Packard has a process to allocate LEX id's and token numbers to users submitting programs to the Users' Library or burning application ROM's.

In addition to having a unique LEX id/token number, the LEX file must have a unique character set id. This number identifies which character set LEX file is active. Theoretically, up to 256 character set LEX files may be present in memory if they each have unique character set id's. It is probably a good idea to have the character set id match the LEX id if possible.

For details about how the LEX file implements the character set, see the HP-71 IDS Volume I.

# 71-00007 SYSTEM MODIFICATIONS

**GENERAL FEATURES**

Alternate character set ______

ASSIGN # ______

ENDLINE ______

EXACT ______

Files ______

**FLAGS**

BEEP ON/OFF ______

Beep volume ______

Math Exceptions ______

OPTION BASE/ROUND/ANGLE ______

Other system or user flags (include flag number) ______

**STARTUP**

Variables ______

Other ______

**DISPLAY**

CONTRAST ______

DELAY ______

FIX/SCI/ENG/STD ______

WIDTH ______

WINDOW ______

**KEYBOARD**

LC ______

Re-defined keys ______

USER mode ______

**HPIL**

ASSIGN IO ______

DISPLAY IS ______

PRINTER IS ______

PWIDTH ______

STANDBY ______

**NOTES** This program does not modify any general features, flags, start up, display, keyboard or HPIL parameters.

# 71-00007

```
10 ! CHARSET - Written by Bruce Stephens
20 ! Creates a LEX file that contains the current character set and adds
     a keyword
30 ! that enables the character set.
40 DIM F$[8],V$[7],W$[8]
50 DESTROY ALL
60 INPUT "New LEX file name: ";F$
70 INPUT "VER$ string: ";V$
80 INPUT "Name of new keyword: ";W$ @ W$=UPRC$(W$) @ IF LEN(W$)<2 THEN 80
90 INPUT "Lex id #(decimal): ";L1
100 INPUT "Token #(decimal): ";L2
110 INPUT "Buffer id #(decimal): ";L3
120 C=LEN(CHARSET$)/6
130 T=121+LEN(V$&W$)+6*C
140 DIM P$[T*2],Z$[C*12+16]
150 P$=FNS$(CHR$(L1))&FNS$(CHR$(L2)&CHR$(L2))
160 P$=P$&"00000f71000000"
170 P$=P$&FNS$(CHR$(20+2*LEN(W$)))
180 P$=P$&"000000"&FNS$(CHR$(144+2*LEN(W$&V$)))
190 P$=P$&"000D"&FNH$(2*LEN(W$)-1)&FNT$(W$)&FNS$(CHR$(L2))
200 P$=P$&"1ff969d031bf961"&FNS$(CHR$(40+LEN(V$)*2))&"0012b1351121C"
210 P$=P$&FNH$(LEN(V$)*2+1)&"1378b6ce13510b3"
220 P$=P$&FNH$(LEN(V$)*2+1)&FNS$(V$)
230 P$=P$&"0215d"&FNH$(LEN(V$)*2+1)
240 P$=P$&"0032bf38f1c8115e23010e0290a2217414b31"
250 P$=P$&FNS$(CHR$(L3))
260 P$=P$&"966311c47e501c330b15d000038d30350"
270 P$=P$&"9ffff2ffffd2307d532bfb8fd79114908d84a807e1017431"
280 P$=P$&FNS$(CHR$(L3))&"14d1cb30115d050e071450375ff"
290 P$=P$&FNH$(C*12)&FNH$(C*12 DIV 16)&FNH$(C*12 DIV 256)
300 P$=P$&FNT$(CHARSET$)
310 CREATE TEXT F$,(LEN(P$)+1) DIV 2          ! Create a file of proper size
320 A=HTD(ADDR$(F$))
330 POKE DTH$(A+37),P$                        ! Poke hex code into file
340 POKE DTH$(A+16),"802e00"                  ! Change file type to LEX
350 STOP
360 DEF FNH$(N)=DTH$(N)[5,5]
370 DEF FNS$(S$)
380 Z$=""
390 FOR Z=1 TO LEN(S$)
400 Z$=DTH$(NUM(S$[Z,Z]))[5,5]&DTH$(NUM(S$[Z,Z]) DIV 16)[5,5]&Z$
410 NEXT Z
420 FNS$=Z$
430 END DEF
440 DEF FNT$[1536](S$)
450 Z$=""
460 FOR Z=1 TO LEN(S$)
470 Z$=Z$&DTH$(NUM(S$[Z,Z]))[5,5]&DTH$(NUM(S$[Z,Z]) DIV 16)[5,5]
480 NEXT Z
490 FNT$=Z$
500 END DEF
```

# 71-00008 PROGRAM DESCRIPTION

Program Title Customization Utilities (CUSTUTIL)

Contributor Hewlett Packard Company

Address 1000 N.E. Circle Blvd.

City Corvallis State Or. Country U.S.A.

Telephone (503)757-2000 Zip/Postal Code 97330

Program Description (include equations) CUSTUTIL provides six keywords that are helpful in customizing the user interface: INLINE gives an enhanced input capability; it allows you to determine the cursor position and type, and which keys terminate MSG$ allows for localization of error messages and user input, making it possible for a Basic program to be translated in to any language automatically. KEYWAIT$ puts the 71 in a low power state, waiting for a key to be hit, then returns key name. SCROLL scrolls the message in the display the specified number of characters. KEYNAM$ returns keyname, given keycode. KEYNUM returns keycode, given keyname.

Necessary Accessories None

Operating limits and warnings

Minimum RAM Requirement 1007 bytes

References

Syntax and Explanation of each Keyword:

## INLINE Statement

Syntax:

```
INLINE <input string>,<char# in LCD position 1>,...
 ...<cursor position/type>,<terminators>,<variable1> [,<variable2>...
 ...[,<variable3>] ]
```

**<input string>::=** String expression to be displayed as prompt

**<char# in LCD pos. 1>::=** Numeric expression which rounds to X,
such that: 1<= X <= 96
Value out of range generates error.

Determines how many characters of displayed string are scrolled off left end of the display. For example:
1=> no characters scrolled
2=> 1 character scrolled

**<cursor pos/type>::=** Numeric expression which rounds to X,
such that: 1<= |X| <=96
Value out of range generates error.

Determines which character in the display the cursor is on. Regardless of input, this value is forced to be at least as large as the char# of the first readable character in the display; also, it is forced to be no bigger than 1 character position beyond the last readable character in the input string.

Negative argument indicates an insert cursor.

**<terminators>::=** String expression of the form:
#<physical keycode>#<physical keycode>...

Keys are numbered in row-major order 1-56. For f-shifted keys, add 56; for g-shifted keys, add 112.

Determines which keys terminate INLINE. Null string or string not conforming to syntax above generates error.

'#' as last character in string is ignored.

**<variable1>::=** Numeric variable into which the terminator number is returned. The variable specified contains 'n' on exit if the terminator hit was the nth specified in the terminator list.

**<variable2>::=** Numeric variable into which the final cursor position and type is returned.

Assuming |variable2| = n, the cursor was on the nth character in the 'free portion' of the display buffer. See the discussion of WINDOW in the HP-71 Reference Manual for details.

If <variable2> < 0 , then insert cursor

<variable3>::= Numeric variable into which the character# in LCD position 1 is returned.

Once again, note that WINDOW affects the effective size and location of the LCD.

Description:

INLINE is a statement that extends the capability given in the HP-71's INPUT statement and KEY$ function. INLINE allows you to specify
- a) the prompt string
- b) the number of prompt string characters to be scrolled off the left side of the display
- c) where in the display the cursor is to come up flashing, and
- d) what type of cursor (replace/insert)

INLINE allows the user to press any combination of keys for input and editing, just like the INPUT statement. While INPUT terminates execution only when specific keys are pressed (such as [Endline]), any number of different keys can be defined to terminate INLINE execution. When one of these terminating keys is pressed, INLINE returns a number that indicates which key caused termination; INLINE will optionally return additional values indicating the cursor position/type and number of characters scrolled off the left side of the display on exit.

For increased customization, the input string may contain cursor on and cursor off characters to make certain portions of the string non-editable.

There are three additional limitations placed on the input parameters for <char# in LCD pos. 1> and <cursor pos.>:
1) If <char# in LCD pos 1> is greater than <cursor pos>, then <char# in LCD pos 1> is set equal to <cursor pos>.
2) <char# in LCD pos 1> is limited to be <= 97 - WINDOWsize
3) If <cursor pos> exceeds <char# in LCD pos 1> + WINDOWsize, then the specified <cursor pos> takes precedence, and the <char# in LCD position 1> is incremented until the 'cursor character' appears in the display window.

For example,

```
INLINE A$,91,80,T$,A
```

According to (1) above, <char# in LCD pos 1> becomes 80, instead of 91. Then, according to (2) above, <char# in LCD pos 1> becomes 75 (assuming the default WINDOWsize of 22).

To illustrate (3) above

```
INLINE A$,60,95,T$,A
```

In order to get character #95 in the display window, character #74 (96-22) is put in LCD position 1.

Following is an example illustrating the use of protected fields (non-editable characters) in the <input string>:

```
INLINE CHR$(27)&"<"&"Enter Name "&CHR$(27)&">"&C$,2,1,"#38#50#51",A,B,C
```

Assume that C$ contains the default input string. In this example the user cannot back the cursor up over the prompt since the cursor was turned off. However, they can edit the default input string since the cursor was turned back on. The replace cursor will come up on the first 'readable' character, that is the first character displayed in which the cursor is on (in this example that is the first character of the default input string) -- this was specified by the cursor position/type argument. The first character of the input string will be scrolled off the left side of the display -- this was specified by the next argument.

INLINE will terminate on one of three keys: [Endline],[Up arrow],[Down arrow]. If [Down arrow] is the terminator key, A=3 on exit. If the user typed in a five character name before hitting the terminator key (assuming no backspaces), B=17 on exit (the cursor originally came up on the 12th character in the display and was advanced 5 more character positions), and C=2.

Note that the <cursor position> argument 'counts' readable characters only. Also, DISP$ 'sees' readable characters only, so that a DISP$ done in the above example returns only the user input (including the default input), not the prompt itself.

Also note that the cursor position argument and the value returned in the first optional variable do not operate totally analogous. The cursor position argument counts readable characters only, whereas the value returned in B (in the example above) reflects the TOTAL number of characters in the "free portion" of the display, readable and non-readable.

Related Keywords:

DISP$, WINDOW

KEYNAM$ Function
-------

Syntax:

KEYNAM$(<physical keycode>)

<physical keycode>::=    Numeric expression, rounded to integer X, such that 1<= X <=168

All values out of range (with the exception of zero) generate an error. KEYNAM$(0) returns the null string.

Description:

Given the physical keycode (keys are numbered in row-major order), KEYNAM$ returns the corresponding key name. Refer to the KEY$ function in the HP-71 Reference Manual for an explanation of key names.

KEYNAM$ is the complement of KEYNUM.

Examples:

```
KEYNAM$(1)   -- returns Q
KEYNAM$(113) -- returns q
KEYNAM$(57)  -- returns fQ
```

Related Keywords:

KEYNUM

KEYNUM Function
------

Syntax:

KEYNUM(<key name>)

<key name>::= String expression

Any string that isn't a valid key name generates an error, with one exception: If the string is null, KEYNUM returns 0.

Refer to 'key name' in the glossary of the HP-71 Reference Manual for further details.

Description:

Given a key name, KEYNUM returns the corresponding physical keycode. It is the complement of KEYNAM$.

Examples:

```
KEYNUM("Q")     -- returns 1
KEYNUM("fQ")    -- returns 57
KEYNUM("#113") -- returns 113
```

Related Keywords:

KEYNAM$

KEYWAIT$ Function
--------

Syntax:

KEYWAIT$

Description:

When the KEYWAIT$ function is executed, the HP-71 goes into a low power consumption state until a key is pressed; when a key is pressed, KEYWAIT$ returns the corresponding key name.

Related Keywords:

KEY$

MSG$ Function
----

Syntax:

MSG$(<lllmmm>)

where lll is the three-digit LEX file ID
  and mmm is the three-digit message number.

If the specified LEX file doesn't exist, or if the specified message number does not exist in the LEX file, MSG$ returns the null string.

Description:

MSG$ allows a BASIC user to build custom messages from any message table. In addition, the translation capability provides a powerful tool for BASIC application pacs to accept commands in any language. An excellent example is the HP-71 Text Editor, a BASIC program that stores all its commands, responses, and HELP catalog information in a message table. All user input is compared to entries in the message table, using MSG$.

To build your own foreign language LEX file, refer to MSG$ in the HP-71 IDS Volume I.

Examples:

```
DISP MSG$(255131)  -- displays message number 131 from LEX file 255,
                      according to the foreign language LEX file that
                      is currently plugged in.

DISP MSG$(085001)  -- displays the first message from LEX file 85
```

SCROLL Statement
------

Syntax:

SCROLL <char# in LCD pos. 1>

<char# in LCD pos. 1>::= Numeric expression, rounded to an integer value.
Errors if negative.

Description:

The SCROLL statement scrolls the message in the display the necessary number of characters, so that the character you specify appears in LCD position 1.

The number of characters can be specified by any positive numeric expression. An error results if the rounded integer value is negative, or if it exceeds 1,048,575 (FFFFF Hex).

For a rounded integer value of 0, SCROLL interprets the parameter as 1.

Related Keywords:

WINDOW

# 71-00009 PROGRAM DESCRIPTION

Program Title Extended Showport

Contributor HEWLETT PACKARD COMPANY

Address 1000 N.E. Circle Blvd.

City Corvallis State OR. Country U.S.A.

Telephone (503)757-2000 Zip/Postal Code 97330

Program Description (include equations) SHOWPORT in operating release 1BBBB only gives information on RAM which has been freed with FREEPORT. This lexfile extends SHOWPORT so it gives information on all RAM.

After SHOWPORT gives information on all independent RAM and ROM, it gives the information on all other RAM (system RAM). The device type number of system RAM is 0.

The extended SHOWPORT lexfile is operated from the keyword "Showport".

Necessary Accessories None

Operating limits and warnings

Minimum RAM Requirement 151 bytes

References

# 71-00010 PROGRAM DESCRIPTION

Program Title Simple and Enhanced Key Redefinition

Contributor HEWLETT-PACKARD COMPANY

Address 1000 NE Circle Blvd

City Corvallis State Oregon Country U.S.A.

Telephone (503) 757-2000 Zip/Postal Code 97330

Program Description (include equations) KEYDEF allows keys to be redefined with a minimum of keystrokes. It leads the user through the redefinition process with a straightforward series of prompts. The user can also choose to scroll through the "keys" file, viewing and editing already-existing key assignments. It also provides a simple mechanism for imbedding escape characters in an assignment string using an intuitive list of mnemonics (see page 10).

Necessary Accessories CUSTUTIL LEX file

Operating limits and warnings

Minimum RAM Requirement 3214

References HP-71 Owners Manual, Section 7 - Redefining the Keyboard

HP-71 Reference Manual - DEF KEY

# 71-00010 VARIABLE DEFINITIONS

| NAME | DEFINITION |
|---|---|
| C | Cursor position in user-input string. |
| C2 | Cursor position while entering escape character. |
| E | Ending Keycode - largest physical keycode that has an assignment associated with it. |
| F | Flag indicates keys file was secure on entry. |
| I | Index variable for scrolling through keys file - contains physical keycode. |
| J | Index variable for matching user-input escape sequence mnemonic to corresponding escape character. |
| K | Indicates which key terminated user input. |
| L | Character # in LCD position 1 (for INLINE prompting). |
| P | Position of blank in D$; Position of escape character in assignment string. |
| S | Starting keycode - smallest physical keycode that has an assignment associated with it;<br>= -1 if not yet determined<br>= Ø if no redefined keys |
| W | Window start - ensures prompt is in protected field of display. |
| A$ | Assignment string currently (or proposed to be) associated to a particular key. |
| D$ | Display contents when scrolling through key assignments. |
| E$ | Array of escape sequence mnemonics, and their corresponding escape characters. |
| E1$ | Escape sequence mnemonic input by user. |
| K$ | Indicates key to redefine. |
| P$ | Prompt. |
| R$ | User response to "Y/N" prompt. |
| T$ | Type of assignment currently - ":", ";", or space. |
| T1$ | Type of assignment proposed - " " |
| Z$ | Saves information about the user's environment:<br>Z$[1,6] - 1st alternate character set character<br>Z$[7,21] - System flags -13 through -64 User flags 0 through 7 |

# 71-00010 SAMPLE PROBLEM

Assume you want keys redefined as follows:

The [Q] key is to become a typing aid to display:

A$=A$&A$@

The [RUN] key is to remain a "direct execute" key, in the sense that pushing it will cause execution, without altering the display, but instead of running current file it will

EDIT NEW

The [<] key is to become a typing aid. When hit in User mode, the following will be added to the display contents, and then the entire display contents will be executed as though [END LINE] was pressed

CAT ALL

Additionally, redefine [↑] so that when it is pressed in User mode, some escape sequences are sent to the display device. Have it display ABC, home the cursor, then display DEF.

Re-define [B], then delete the key redefinition.

Finally, before exiting the program, scroll through your file of key redefinitions. Make some modifications and delete a key redefinition.

# 71-00010 SAMPLE PROBLEM SOLUTION

| DISPLAY CONTENTS | USER RESPONSE | COMMENTS |
|---|---|---|
| > | RUN KEYDEF | |
| Define new keys? | Y | |
| Hit key to re-define | Q | |
| String | A$=A$&A$@ [ENDLINE] | |
| Type: ; or : or [SPC] | ; | Hit [g] [=] The ; terminator makes this a typing aid that remains in the display. |
| Assignment complete | | No response |
| Done? | N | |
| Hit key to re-define | [RUN] | Hit the [RUN] key |
| String | EDIT NEW [ENDLINE] | |
| Type: ; or : or [SPC] | : | Hit [g] [*] The colon terminator makes this a direct execute key in user mode, that does not alter the display. |
| Assignment complete | | No response |
| Done? | N | |
| Hit key to re-define | [RUN] | Let's double check |
| String EDIT NEW | [ENDLINE] | KEYDEF shows any string alread assigned. Hitting [ENDLINE] here leaves the assignment unchanged. |
| Type: : ; [SPC] | : | This time ":" displayed first - whichever terminator appears first is the current terminator. |

# 71-00010 SAMPLE PROBLEM SOLUTION

| DISPLAY CONTENTS | USER RESPONSE | COMMENTS |
|---|---|---|
| Assignment complete | | No response |
| Done? | N | |
| Hit key to re-define | < | Hit [g] [.] |
| String | CAT ALL [ENDLINE] | |
| Type: ; or : or [SPC] | [SPC] | Hit the [SPC] key |
| Assignment complete | | No response |
| Done? | N | |
| Hit key to re-define | < | Let's double check this one, too |
| String CAT ALL | [ENDLINE] | No change |
| Type: [SPC] or ; or : | [SPC] | Note that this time [SPC] was the first terminator type displayed |
| Assignment complete | | No response |
| Done? | N | |
| Hit key to re-define | [↑] | Hit the [↑] key |
| String | ABC [RUN] | Hitting the [RUN] key puts th program in the proper mode to recognize escape sequence mnemonics. Notice that after [RUN] is hit, the 0 annunciator comes on, indicating it is waiting for a mnemonic. |

# 71-00010 SAMPLE PROBLEM SOLUTION

| DISPLAY CONTENTS | USER RESPONSE | COMMENTS |
|---|---|---|
| String ABC | CHM [RUN] | User enters "cursor home" mnemonic. Hitting the [RUN] key a second time toggles out of the mnemonic mode and turns off the 0 annunciator. |
| String ABC$^{E}$cH | DEF [ENDLINE] | See note below (*) |
| Type ; or : or [SPC] | ; | Hit [g] [=] |
| Assignment complete | | No response |
| Done? | N | |
| Hit key to re-define | [B] | Let's re-define the [B] key, then delete the key re-definition. |
| String | BBB [ENDLINE] | |
| Type: ; or : or [SPC] | ; | |
| Assignment complete | | [B] re-defined |
| Done? | N | |
| Hit key to re-define | [B] | |
| String BBB | [f] [RUN] | Delete the key re-definition |
| Assignment deleted | | |
| Done? | Y | Done re-defining new keys |
| Scroll thru keys? | Y | |
| Initializing KEYSCROLL | | This takes about 12 seconds |

# 71-00010 SAMPLE PROBLEM SOLUTION

| DISPLAY CONTENTS | USER RESPONSE | COMMENTS |
|---|---|---|
| KEY Q ;A$=A$&A$@ | [↓] | There is nothing wrong with th program! It takes about 6 seconds to display the next ke assignment, since there are nc re-defined keys between Q (Keycode #1) and [RUN] (keycod #46). KEYSCROLL checks each key to see if it's redefined |
| KEY #46 :EDIT NEW | [↓] | |
| KEY #50 ;ABC$^E$c HDEF | [↓] | It takes about 14 seconds to see the next re-defined key, since it has keycode 166. The program operates much more rapidly when redefined keys have keycodes that are closer together. |
| KEY < CATALL | [g] [↑] | Go to first re-defined key |
| KEY Q ;A$=A$&A$@ | [→] [→] [f] [-line] [RUN] | Key assignments can be changed as well as viewed from KEYSCROLL |
| KEY Q ;A$ | CFL [RUN] | Cursor far left mnemonic |
| KEY Q ; A$$^E$c ∝ | [ENDLINE] | |
| Assignment complete | | Next re-defined key displayed automatically |
| KEY #46 : EDIT NEW | [g] [↓] | Go to last key re-definition |
| KEY < CAT ALL | [←] [;] [ENDLINE] | Change the terminator type |
| Assignment complete | | |

# 71-00010 SAMPLE PROBLEM SOLUTION

| DISPLAY CONTENTS | USER RESPONSE | COMMENTS |
|---|---|---|
| Key< ; CAT ALL | [f] [RUN] | then, decide to delete the key re-definition |
| Assignment deleted | | |
| KEY #50 ;ABC$^{E}$cHDEF | [ATTN] | Exits KEYSCROLL |
| Define new keys? | N | |
| Scroll thru keys? | N | |
| Exited KEYDEF | | |

(*) Note that the key assignment above could have been handled a bit differently:

| DISPLAY CONTENTS | USER RESPONSE | COMMENTS |
|---|---|---|
| Hit key to re-define | [↑] | |
| String | ABCDEF [←] [←] [←] [RUN] | Type in entire ascii string, position to proper spot in string, then toggle into mnemonic mode. Note that while the 0 annunciator is on, the cursor keys are disabled. |
| String ABCDEF 0 | CHM [RUN] | |
| String ABC$^{E}$cHDEF | [ENDLINE] ⋮ | |

Miscellaneous notes:

Hitting [ATTN] when the 0 annunciator is lit, automatically takes the program out of mneumonic entry mode.

In the scrolling portion of the program,to avoid ambiguity [SPC], f [SPC], and g [SPC] ar represented by their key numbers : #49, #105, #161 respectively.

If KEYDEF is interrupted via the ATTN key, and never allowed to exit normally, the following may be changed from what they were on entry:

71-00010

**GENERAL FEATURES**

Alternate character set: The first alternate character (CHR$(128)) is set to $^{E}C$ (CHR$(31) & CHR$(21) & CHR$(113) & CHR$(80) & CHR$(80))

ENDLINE

EXACT

Files: If the user answers 'Y' to the prompt asking to 'Unsecure keys file', and suspends the program, the keys file will still be unsecure. When the program exits normally*, it re-secures the keys file, and gives a message to that effect.

**FLAGS**

BEEP ON/OFF

Beep volume

Math Exceptions

OPTION BASE/ROUND/ANGLE

Other system or user flags (include flag number): Flag - 16 (Option Base is set to 0)
Flags 0,5

**STARTUP**

Variables: If KEYDEF is suspended by ATTN and not to be continued, then entering END from the keyboard will restore all your variables (Executing END will not restore CHR$(128), the status of the keys file, or flags 0, 5, -16).

Other

**DISPLAY**

CONTRAST

DELAY

FIX/SCI/ENG/STD

WIDTH

WINDOW: Window is changed to 1 (machine default)

**KEYBOARD**

LC

Re-defined keys: Whatever the user changes them to

USER mode

**HPIL**

ASSIGN IO

DISPLAY IS

PRINTER IS

PWIDTH

STANDBY

**NOTES** *It is perfectly acceptable to interrupt KEYDEF using the ATTN key. However, the only way to restore your system to its previous state is to CONT; this gives KEYDEF the opportunity to restore your variables, CHR$(128), flags 0, 5, -16, etc. You know KEYDEF has done this when it gives the message "Exiting KEYDEF". If ATTN is hit during a prompt requiring a "Y" or "N" response, then when the program continues, the prompt is not

# 71-00010 SAMPLE PROBLEM

| Mnemonic | Escape Character | | Effect |
|---|---|---|---|
| INSW | | N | Insert cursor (with wrap-around) |
| INS | | Q | Insert cursor |
| RPL | | R | Replace cursor |
| CRT | | C | Moves cursor right |
| CLT | | D | Moves cursor left |
| CHM | | H | Homes cursor |
| CD | | J | Clears display |
| DEL | | K | Deletes through end of line |
| CON | | > | Turns cursor on |
| COFF | | < | Turns cursor off |
| RD | | E | Resets display |
| DCW | | O | Deletes character (with wrap-around) |
| DC | | P | Deletes character |
| CPV | | % | Sets cursor position in video monitor (See page 328 HP-7 Reference Manual) |
| CFR | (CHR$(3)) | ← | Moves cursor to right of righmost character |
| CFL | (CHR$(4)) | | Moves cursor to leftmost character |

# PROGRAM DESCRIPTION



Program Title ROMAN 8 Character Set Lexfile

Contributor Hewlett-Packard Company

Address 1000 NE Circle Blvd

City Corvallis State Oregon Country U.S.A.

Telephone Zip/Postal Code 97330

Program Description (include equations) Lex file adds one keyword: ROMAN8$. This is a string function of no parameters that returns the 768-byte string needed to define ROMAN 8 as the alternate character set with the HP-71 CHARSET command. To define ROMAN 8 as the alternate character set, simply execute:

CHARSET ROMAN8$

The CHARSET command (explained in the HP-71 Reference Manual) allows the user to specify the display bit-patterns that are used to represent the characters 129-255 in the display. ROMAN8$ supplies the bit-patterns for this character set.
The ROMAN 8 character set is supported by many printers and is a standard for foreign-language localization of American products.
While this lex file has no special memory requirements, consideration of memory usage is important. The overhead associated with string manipulation requires that 776 bytes be available whenever the ROMAN8$ keyword is invoked. And in the worst

ROMAN8

Necessary Accessories None

Supported Accessories N/A

Operating limits and warnings None

File name(s) ROMAN8LX

Size of file(s) 850 bytes Additional RAM Requirement to run the program None

References

71-00011

(Continuation Page)

case (if there is currently no alternate character set defined), executing CHARSET ROMAN8$ will take up 772 additional bytes for the newly created charset buffer. So 1548 bytes is required for successful execution of CHARSET ROMAN8$, although only 772 of those are permanently used. (The lex file does not have to stick around after the alternate character set is defined, although it is needed again if the ROMAN 8 character set is desired after the alternate character set has been redefined.)

71-00011

**SAMPLE PROBLEM**

Sample Usage: CHARSET ROMAN8$

## ROMAN 8 CHARACTER SET

| | | |
|---|---|---|
| CHR$(161)='À' | CHR$(193)='ê' | CHR$(225)='Ã' |
| CHR$(162)='Â' | CHR$(194)='ô' | CHR$(226)='ã' |
| CHR$(163)='È' | CHR$(195)='û' | CHR$(227)='Ð' |
| CHR$(164)='Ê' | CHR$(196)='á' | CHR$(228)='ð' |
| CHR$(165)='Ë' | CHR$(197)='é' | CHR$(229)='Í' |
| CHR$(166)='Î' | CHR$(198)='ó' | CHR$(230)='Ì' |
| CHR$(167)='Ï' | CHR$(199)='ú' | CHR$(231)='Ó' |
| CHR$(168)='´' | CHR$(200)='à' | CHR$(232)='Ò' |
| CHR$(169)='`' | CHR$(201)='è' | CHR$(233)='Õ' |
| CHR$(170)='ˆ' | CHR$(202)='ò' | CHR$(234)='õ' |
| CHR$(171)='¨' | CHR$(203)='ù' | CHR$(235)='Š' |
| CHR$(172)='˜' | CHR$(204)='ä' | CHR$(236)='š' |
| CHR$(173)='Ù' | CHR$(205)='ë' | CHR$(237)='Ú' |
| CHR$(174)='Û' | CHR$(206)='ö' | CHR$(238)='Ÿ' |
| CHR$(175)='£' | CHR$(207)='ü' | CHR$(239)='ÿ' |
| CHR$(176)='¯' | CHR$(208)='Å' | CHR$(240)='Þ' |
| CHR$(177)=' ' | CHR$(209)='î' | CHR$(241)='þ' |
| CHR$(178)=' ' | CHR$(210)='Ø' | CHR$(242)=' ' |
| CHR$(179)='°' | CHR$(211)='Æ' | CHR$(243)=' ' |
| CHR$(180)='Ç' | CHR$(212)='å' | CHR$(244)=' ' |
| CHR$(181)='ç' | CHR$(213)='í' | CHR$(245)=' ' |
| CHR$(182)='Ñ' | CHR$(214)='ø' | CHR$(246)='–' |
| CHR$(183)='ñ' | CHR$(215)='æ' | CHR$(247)='¼' |
| CHR$(184)='¡' | CHR$(216)='Ä' | CHR$(248)='½' |
| CHR$(185)='¿' | CHR$(217)='ì' | CHR$(249)='ª' |
| CHR$(186)='¤' | CHR$(218)='Ö' | CHR$(250)='º' |
| CHR$(187)='£' | CHR$(219)='Ü' | CHR$(251)='«' |
| CHR$(188)='¥' | CHR$(220)='É' | CHR$(252)='■' |
| CHR$(189)='§' | CHR$(221)='ï' | CHR$(253)='»' |
| CHR$(190)='ƒ' | CHR$(222)='β' | CHR$(254)='±' |
| CHR$(191)='¢' | CHR$(223)='Ô' | CHR$(255)='' |
| CHR$(192)='â' | CHR$(224)='Á' | |

Mass-storage Catalog
01/23/85 19:04:34

Volume label:

| NAME | S | TYPE | LEN | DATE | TIME |
|---|---|---|---|---|---|
| ROWCOL | | LEX | 119 | 01/01/00 | 00:56 |
| LIFELEX | | LEX | 457 | 01/01/00 | 00:57 |
| BANNER | | LEX | 202 | 01/01/00 | 00:57 |
| CLOCKDSP | | LEX | 328 | 01/01/00 | 00:58 |
| TEXTUTIL | | LEX | 1512 | 01/01/00 | 00:58 |
| CHARSET | | BASIC | 1328 | 01/01/00 | 00:59 |
| CUSTUTIL | | LEX | 1007 | 01/01/00 | 01:01 |
| SHOWPORT | | LEX | 151 | 01/01/00 | 01:01 |
| KEYDEF | | BASIC | 3214 | 01/01/00 | 01:02 |
| ROMAN8LX | | LEX | 850 | 01/01/00 | 01:04 |