Joseph K. Horn

PPC Paris

JPC Rom

Owner's Manual

April 1988
Revision D

FOREWORD

This manual is a reference tool for JPC Rom users. It is assumed that the users are already familiar with HP-71 operation and Basic language programming.

JPC Rom combines over 100 keywords into a single 25K Lex file. This brings together some of the most useful HP-71 enhancements and improves the overall performance of the machine. The source material was contributed by members of the international user community, most of whom are members of PPC Paris.

All contributions have been made in the same spirit of benevolent, cooperative, mutual assistance.

The manual itself was an enormous task. Any remarks or comments about its contents are welcome. It was carried out by Pierre David, Jean-Jacques Dhénin and Janick Taillandier. Special thanks are due to Michael Markov for his help during the translation of this manual and for his support of our work.

It is also worth noting that the printing was done with an HP-71 and a LaserJet printer.

We hope you will appreciate the result. Don't hesitate to give us your feelings :

PPC Paris
B.P. 604
75028 Paris Cedex 01
France

Distributed by:
Corvallis Microtechnology
895 NW Grant Ave
Corvallis, OR 97330
TEL: (503)752-5456

MAIN DIFFERENCES BETWEEN VERSION A AND VERSION B

The differences between version A00 and version B00 (see VER$ in *Other JPC Rom features*) result from some bug fixes, modifications and improvements. The main differences are listed below :

Bug fixes

MARGIN parameter is now limited to 96.

Under certain circumstances FIND could find matching lines which should not have been. (see *JPC 45*). Janick Taillandier has corrected this problem.

COMB and ARR have been rewritten by Guy Toublanc to use a multiplication based algorithm instead of factorials.

COMB was modified to return a valid result when executing S=S+COMB(n,0).

The FINPUT version published in *JPC* exited when pressing the [f] [CONT] key. This was wrong.

New keywords and features

The DATESTR$ function has been added to convert from and to the new date format.

New structured programming statements have been added to this version.

Modifications and improvements

Date functions have been modified to use the new JPC Rom date format as well as the standard format.

The keyword KSPEED has been removed. The cursor speed-up is still present, but repetition speed is maximum and delay after the key is pressed and before repeat begins is not tunable.

POSI has been modified to admit numeric as well as string parameters.

The escape sequence sent to the printer by BOLD has been modified to be compatible with all *PCL* printers, especially the ThinkJet and LaserJet.

FF has been renamed to PFF, LF to PLF, PL to PAGELEN, CR to PCR. FPRM has been renamed to FPRIM and NPRM to NPRIM. HMS+ has been renamed to HMSADD and HMS− to HMSSUB.

Note

All these improvements and corrections have been realized preserving program compatibility with previous versions of JPC Rom. So your programs written with previous versions are totally compatible with the new JPC Rom.

MAIN DIFFERENCES BETWEEN VERSION B AND VERSION C

The differences between version B00 and version C00 (see VER$ in *Other JPC Rom features*) result from some bug fixes, modifications and improvements. The main differences are listed below :

Bug fixes

Structured programming statements didn't work properly if followed by a comment after the beginning of the loop, as for example :

```
        10 WHILE 1 !
        20 END WHILE
```

Same problem with SELECT, CASE, etc. Notified by Henri Kudelski from Switzerland and Gérard Kossman in France.

Functions STARTUP$ and ENDUP$ didn't returned their result properly, this could produce a Data type error when the following program was executed :

```
        10 DESTROY ALL
        20 DIM S$
        30 DIM S$[LEN(STARTUP$)]
```

Same problem with ENDUP$. Notified by Tapani Tarvainen from Finland.

The [f] [BACK] key acts as usual in the Command Stack under CALC mode, this allows freely editing expressions in the command line. Notified by Michael Markov in the United States and Tapani Tarvainen in Finland.

After a configuration (for example, COPY of a Lex file to Ram, LEX ON/OFF), assembler tabs were enabled when using EDTEXT.

The keyword STACK has been replaced by a new version from Henri Kudelski in Switzerland to avoid bad problems during the ML program.

New keywords and features

The keywords SYSEDIT, OPCODE$ and NEXTOP$ have been added. OPCODE$ and NEXTOP$ were written by Jean-Jacques Dhénin. SYSEDIT was written by Pierre David and Janick Taillandier.

The FILESIZE function by Henri Kudelski has been added.

The address directory manager KA and its related programmable functions (ADCREATE, ADDELETE, ADFIND, ADGET, ADPUT and ADSIZE) have been added. These keywords have been written by Pierre David.

The KEYWAIT$ function by Hewlett-Packard has been added. Its Id and token numbers have not been modified versus the Users' Library version.

The keyword ROMAN was added to allow using the *Roman* character set. This keyword was written by Pierre David and Janick Taillandier.

Now, JPC Rom recognizes non standard file types, such as HP-41 or HP-75 ones, during a CAT, as well as non standard HP-71 types. This was written by Jan Buitenhuis in The Netherlands and Janick Taillandier in France.

Modifications and improvements

JPC Rom was previously called *JPCLEX*.

BLIST has been renamed to DBLIST, because of a conflict with the BREAK Lex from the Users' Library.

SWAP has been renamed to VARSWAP.

The INV$ function has been removed, its functionalities are now part of INVERSE.

Syntax of INVERSE and PAINT have been extended to provide more flexibility.

Syntax of SPACE$ has been extend to allow repeating any string.

ENABLE and DISABLE have been renamed to LEX ON/OFF because of a conflict with ENABLE in the HP-IL module.

Functions REPLACE$ and RPLC$ have been merged in a new REPLACE$. With three parameters or if the fourth one is numeric, functionalities are similar to the old RPLC$. If the fourth parameter is a string it is the *wild-card* character used in the old REPLACE$.

DBLIST and PBLIST were rewritten to allow indentation of structure and redirection into a file.

Removed keywords are listed as obsolete when they are present in a program. If these programs are executed, they produce the error JPC ERR:Removed Keyword (message number 16).

Note

All these improvements and corrections have been realized preserving program compatibility with previous versions of JPC Rom. So your programs written with previous versions are totally compatible with the new JPC Rom.

MAIN DIFFERENCES BETWEEN VERSION C AND VERSION D

The differences between version C and version D (see VER$ in *Other JPC Rom features*) result from some bug fixes, modifications and improvements. The main differences are listed below :

Bug fixes

The extended character set *Roman* disappeared at power on.

The *Assembler Tabs* mode was regularly enabled (during power on, for example).

The POSI function returned an incorrect value (1) when used as `POSI("",`*x*`)`, with $x<6$. This bug was mentioned by Joe Horn in the United States.

The new DBLIST version did not recognize the following syntax : `DBLIST 1000 INDENT 4`.

The FIND keyword did not work properly. Notified by Henri Kudelski from Switzerland and Claudio Benski from France.

Date calculation functions (such as DOW for example) did not accept February 29 of leap years, when the last year digit was not a multiple of 4. Notified with details by Laurent Istria from France.

From *browse mode* in KA, the keystroke [f] [EDIT], then [ENDLINE] entered *edit mode*, then exited it. Notified by Henri Kudelski from Switzerland.

New keywords and features

The DDIR and PDIR keywords have been added.

Modifications and improvements

Disassembling with OPCODE$ and SYSEDIT swapped RSI and PC=(A) mnemonics. This appeared when we disassembled the HP-28C Rom.

OPCODE$ → RSI = 8081
PC=(A) = 808C

Note

All these improvements and corrections have been realized preserving program compatibility with previous versions of JPC Rom. So your programs written with previous versions are totally compatible with the new JPC Rom.

NOTE: VERSION "EX", HEREIN REFERRED TO AS "REV X", IS AN UNOFFICIAL IMPROVEMENT TO JPCLEX DONE BY RODGER ROSENBAUM. THE CHANGES ARE INDICATED IN THE APPROPRIATE PLACES. JH

INCLUDED IN JPCROM REV X:

JPCLEX
MYSUBS
HLPLEX
FACTORLX
NIBBLEX

Hi, Gene!

** CHANGES IN JPC ROM **

The enclosed JPC ROM has been greatly modified by Rodger Rosenbaum. He removed all the known bugs (and many unknown bugs), and added much functionality to it. Here are the changes, as best I can remember:

(1) ARR (arrangements) has been renamed to PERM (permutations).
(2) PPCM (?) has been renamed to LCM (least common multiple).
(3) PGCD (?) has been renamed to GCD (greatest common divisor).
(4) LEX ON and LEX OFF have been renamed to LXON and LXOFF to prevent conflicts with other commonly-used lexfiles.
(5) Textfiles can be listed directly.
(6) PRIM(1)=1. (bugfix)
(7) PRINTER IS NULL doesn't crash (bugfix).
(8) In KA, the meaning of "." at the end of a search line has been switched in meaning. Now all searches are generic by default, and if you want a particular search, type a "." at the end of the line.
(9) DBLIST and PBLIST default indentation: changed from 0 to 2.
(10) FIND used to occasionally stop at lines that did not contain the target string. It works correctly now.
(11) EDIT TO has been debugged.
(12) CENTER$ does RED$ first (bugfix).
(13) DDIR and PDIR scroll properly in the LCD (bugfix).
(14) SYSEDIT stays in disassemble mode when pressing +, -, *, /, etc.
(15) SYSEDIT now remembers the current address when it exits, allowing you to continue later by invoking it without an address.
(16) The SCROLL and MSG$ commands (usually used from the KEYWAIT lexfile) have been added to JPCLEX.
(17) PEEK$ and POKE have been deleted from JPCLEX (they exist elsewhere in the ROM already).
(18) The ROM also contains (besides JPCLEX), MYSUBS (my collection of favorite subprograms, as written up in the CHHU Chronicle); HLPLEX (type HELPF HLPLEX to see its contents and see it in action), FACTORLX (faster then PRIM), and NIBBLEX (collection of improved versions of PEEK$, as written up in the CHHU Chronicle).

If you have any questions, please holler. -jkh-

ADBUF$

E1/01

ADBUF$ (buffer address) returns the address of the buffer specified by its identification number.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
ADBUF$ ( buffer id )
```

Example

A$=ADBUF$("BFC") Stores the "lex buffer" address in A$.

Input Parameter

Item	Description	Restrictions
buffer identifier	String expression containing hexadecimal digits.	3 upper or lower case digits.

Operation

Buffers :

HP-71 buffers are used to store volatile (more so than in files) or data used only by the operating system.

They are used by assembly language applications or by the system. The following table lists various buffers used by the system :

```
 Id  Description
----------------------------------------------------
808  Hold a string of characters used by STARTUP
83D  MARGIN setting
83E  Hold a string of characters used by ENDUP
BFB  Character set defined by CHARSET
BFC  Address of Lex files
```

Buffers consist of a 7 nibble header followed by the the data area itself. The header has the following structure :

1 nibble : number of addresses in the beginning of the the buffer that need to be updated when memory moves,
3 nibbles : the buffer ID,
3 nibbles : buffer length in nibbles (data part only).

Buffer are mobile areas. Their address change often, especially when :
- a file is created, deleted or when its size is changed,
- another buffer is created, deleted or when its size is changed.

The ADBUF$ function :

ADBUF$ returns the address of the buffer whose ID is given. If it can't be found a null string is returned. The address returned by the function is the address of the buffer header. Information stored in a buffer is located 7 nibbles further.

References

JPC 22 (page 35) first version of ADBUF$ by Michel Martinet et Pierre David.

JPC 23 (page 30) HP-71 buffers, by Pierre David. Introductory article and various Basic utilities.

JPC 27 (page 34) second version by Michel Martinet.

Internal Design Specification Volume I, Chapter 3.5.3.

Related Keywords

DTH$, HTD, PEEK$, POKE, ADDR$

Authors

Pierre David and Michel Martinet.

ADCREATE

E1/66

The ADCREATE keyword create an empty address file.

■ Statement	■ Keyboard Execution
O Function	O CALC Mode
O Operator	■ IF...THEN...ELSE
	■ Device Operation

```
ADCREATE file
ADCREATE file , password
```

Examples

`ADCREATE ESSAI`	Creates an address file, without password, whose name is ESSAI.
`ADCREATE A$,"passe"`	Creates an address file and sets the password to «passe».

Input Parameters

Item	Description	Restrictions
file	String expression or unquoted string.	Filename with optional device specifier.
password	String expression. Default : No password	8 first characters only are used.

Operation

Address files

KA puts you in an *address directory* interactive mode. KA allows you to store addresses in a file whose filetype is ADRS. Since KA has been designed to be used only in interactive mode, JPC Rom provides an additional set of functions (ADCREATE, ADGET, ADPUT, ADDELETE, ADSIZE and ADFIND) to access stored addresses from a program.

Address files can be considered as a set of index cards, each one containing an address. For example :

ADCREATE (continued)

```
         ------------------
        |          Name     |
        |------------------|
     ------------------    |
    |          Name    |---|
    |------------------| 1 |
 ------------------    |---|
|          Name    |---| 2 |
|------------------| 1 |---|
|          Phone   |---| 3 |
|------------------| 2 |---|
|          Line 1  |---| 4 |
|------------------| 3 |---|
|          Line 2  |---|   |
|------------------| 4 |---|
|          Line 3  |---|   |
|------------------|   |---
|          Line 4  |---|
|------------------|   |
|          Note    |---
|------------------|
|          Index   |
 ------------------
```

In this example, the file contains three cards. Let us examine the card contents.

The cards

Each card is made up of 8 lines, organized as follows :

- name and first name, separated by a /,
- phone number,
- 4 lines to store the address,
- a line to store general informations or comments, and
- a line to store an index to be used by your own programs.

The first line contains the name and first name, separated by a slash (/). The address directory functions will add it for you if you forget to enter the slash.

Address directory management functions

You have 6 functions :

- ADCREATE creates a file with the ADRS filetype ; this function may optionaly specify a password on the file,

- ADGET reads an address (a card) from the file and stores it into a string array,

- ADPUT stores a card into the address file,

- ADDELETE removes a card from the file given its sequence number,

- ADSIZE returns the number of cards in the file,

- and ADFIND looks for a card in the file and returns its sequence number.

It is possible to specify a password with all these functions. If a password has been defined for the file, you must specify it with all functions. If the password is not defined, the parameter is optional and is not used.

ADCREATE keyword

ADCREATE creates an empty address file (with type ADRS) and may optionaly specify a password.

ADCREATE cannot create the file if it already exists ; then it returns : `JPC ERR:File Exists`.

The memory requirements for the address directory can be computed by the following formula :
30,5 bytes + size of all cards

The size of a card can be computed by the following formula :
10 bytes + number of characters in the card

References

Program AGENDA for the HP-75 by Pierre David.

Related Keywords

ADDELETE, ADFIND, ADGET, ADPUT, ADSIZE, KA

Author

Pierre David

ADDELETE

E1/67

ADDELETE removes a card from an address file.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
ADDELETE file , number
ADDELETE file , number , password
```

Examples

`ADDELETE ESSAI,5`	Removes the fifth card in the file ESSAI, without password.
`ADDELETE A$,I+1,P$`	Removes card number I+1 from the address file specified by variable A$ with password specified by P$.

Input Parameters

Item	Description	Restrictions
file	String expression or unquoted string.	The file must be in Ram.
number	Numeric expression rounded to an integer.	Must be between 1 and the number of cards in the file.
password	String expression. Default : No password.	8 first characters only are used.

Operation

The keyword ADDELETE removes from the address file the card whose sequence number is specified.

ADDELETE cannot delete the card if :
- the file is not in Ram,
- the filetype is not ADRS,
- the file contains a password and the password specified by the keyword is invalid,
- the card number is invalid.

Please refer to keyword ADCREATE for more information about address files.

References

Program AGENDA for the HP-75 by Pierre David.

ADDELETE (continued)

Related Keywords

`ADCREATE`, `ADFIND`, `ADGET`, `ADPUT`, `ADSIZE`, `KA`

Author

Pierre David

ADFIND

E1/68

The ADFIND function looks for a name in an address file.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
ADFIND ( file , string )
ADFIND ( file , string , password )
```

Examples ↓

`A=ADFIND("ESSAI","Dupond",P$)`	Returns the card corresponding to the name "Dupond" in file ESSAI with password P$.
`ADDELETE A$,ADFIND(A$,"Dup.")`	Removes the card corresponding to the the first name beginning with "Dup" in the address file A$.

Input Parameters

Item	Description	Restrictions
file	String expression ~~or unquoted string.~~	The file must be in Ram.
string	String expression.	None.
password	String expression. Default : No password.	8 first characters only are used.

Operation

The ADFIND function returns the number of the card corresponding to the name provided as parameter *string*.

This card number can then be used with ADGET and ADDELETE functions to copy a card in a string array or delete it from the file.

The *string* parameter follows the same rules as the *search mode* of KA :

- Name only : search only on the name. the first matching name is returned whatever may be the first name.

- Name and first name (separated by a /) : search on the name *and* the first name.

- string terminated by a dot : the search is generic. Names do not need to be input completely. The card returned is the first one after the matched string. There is no error if the string is not exactly found.

The search is not case sensitive : upper case and lower case characters are identical.

ADFIND (continued)

See KA for more information about the search.

If ADFIND fails for any reason, a negative number is returned whose absolute value is the error number who causes the failure.

Please refer to the ADCREATE keyword for more informations about address files.

References

Program AGENDA for the HP-75 by Pierre David.

Related Keywords

ADCREATE, ADDELETE, ADGET, ADPUT, ADSIZE, KA

Author

Pierre David

ADGET

E1/69

The ADGET keyword reads a card and stores it into a string array.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
ADGET file , array , number
ADGET file , array , number , password
```

Examples

```
ADGET ESSAI,T$,5
```

Reads card number 5 in file ESSAI and stores it into the string array variable T$.

```
ADGET A$,T$,I+1,P$
```

Reads card number I+1 in address file identified by variable A$ with password P$.

Input Parameters

Item	Description	Restrictions
file	String expression or unquoted string.	The file must be in Ram.
array	String array name.	Must have exactly 8 elements.
number	Numeric expression rounded to an integer.	Must be between 1 and the number of cards in the file.
password	String expression. Default : No password.	8 first characters only are used.

Operation

The ADGET keyword reads the specified card from the address file and stores it into a string array, to be processed by a user program.

Warning : *array* must have exactly 8 elements. Each element must be long enough to store the data. A line holds at most 91 characters.

Sample program using ADGET to print the addresses in the ADRS file :

ADGET (continued)

```
100 F$="ADRS"               ! file name
110 OPTION BASE 1
120 DIM T$(8)[91]
130 FOR I=1 TO ADSIZE(F$)
140   ADGET F$,T$,I
150   PRINT T$(1)           ! name
160   PRINT T$(3)           ! address 1
170   PRINT T$(4)           ! address 2
180   PRINT T$(5)           ! address 3
190   PRINT T$(6)           ! address 4
200 NEXT I
```

ADGET cannot read the card if :
- the file is not in Ram,
- the file type is not ADRS,
- the file contains a password and the password provided is not valid,
- the card number is not valid,
- the *array* has not enough elements,
- one of the card fields is too long to be stored in an array element.

Please refer to the ADCREATE keyword for more informations about address files.

References

Program AGENDA for the HP-75 by Pierre David.

Related Keywords

ADCREATE, ADDELETE, ADFIND, ADPUT, ADSIZE, KA

Author

Pierre David

ADPUT

E1/6A

The ADPUT keyword write a card into an address file.

■ Statement	■ Keyboard Execution
O Function	O CALC Mode
O Operator	■ IF...THEN...ELSE
	■ Device Operation

```
ADPUT file , array
ADPUT file , array , password
```

Examples

`ADPUT ESSAI,T$`	Writes the card stored in T$ into file ESSAI.
`ADPUT A$,T$,P$`	Writes the card T$ into the address file specified by A$ whose password is in P$.

Input Parameters

Item	Description	Restrictions
file	String expression or unquoted string.	The file must be in Ram.
array	String array name.	Must have exactly 8 elements.
password	String expression. Default : No password.	8 first characters only are used.

Operation

The ADPUT statement writes a card into the address file specified by *file*.

The card is stored automatically in alphabetical order.

Warning : *array* must have exactly 8 elements. Each element must be long enough to store the data. A line holds at most 91 characters.

ADPUT cannot store the card if :
- the file is not in Ram,
- file type is not ADRS,
- the file contains a password and the password provided is not valid,
- the *array* has not enough elements,
- one of the card fields is too long to be stored in an array element.
- there is not enough memory.

Please refer to the ADCREATE keyword for more informations about address files.

ADPUT (continued)

References

Program AGENDA for the HP-75 by Pierre David.

Related Keywords

`ADCREATE`, `ADDELETE`, `ADFIND`, `ADGET`, `ADSIZE`, `KA`

Author

Pierre David

ADSIZE

E1/6B

The ADSIZE function returns the number of cards in an address file.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
ADSIZE ( file )
ADSIZE ( file , password )
```

Examples

`N=ADSIZE("ESSAI")`	Stores into the variable N the number of cards in file ESSAI, without password.
`FOR I=1 TO ADSIZE(A$,P$)`	Loops on all cards in file A$, whose password is in P$.

Input Parameters

Item	Description	Restrictions
file	String expression ~~or unquoted string.~~	The file must be in Ram.
password	String expression. Default : No password.	8 first characters only are used.

Operation

The ADSIZE function returns the number of cards found in the specified address file.

If ADSIZE fails for any reason, a negative number is returned whose absolute value is the error number who causes the failure.

ADSIZE cannot return the number of cards if :
- the file is not in Ram,
- the file type is not ADRS,
- the file contains a password and the password provided is not valid,

Please refer to the ADCREATE keyword for more informations about address files.

References

Program AGENDA for the HP-75 by Pierre David.

Related Keywords

ADCREATE, ADDELETE, ADFIND, ADGET, ADPUT, KA

Author

Pierre David

RENAMED <u>PERM</u> (REV. X)

ARR (Arrangements) computes the number of possible different arrangements (permutations) of n items taken p at a time.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
ARR ( n , p )
```

Example

`A=ARR(4,3)` Stores 24 in variable A.

Input Parameters

Item	Description	Restrictions
n	Numeric expression.	Integer between 0 and 10^{12}-1.
p	Numeric expression.	Integer between 0 and n.

Operation

ARR (n,p) = A_n^p = n! / (n-p)!

ARR (*n*, *p*) compute the number of possible different arrangements (permutations) of n items taken p at a time. This function is very useful in probability and statistics.

In order to increase the range of valid parameters, and to improve accuracy, ARR uses a multiplication based algorithm instead of factorials. This results in long execution times for large numbers.

BUT ARR(84,84) ≠ FACT(84)
also 102, 103, 150, 159, 163, 182, 190, 193, 208, 209, 214, 230

In every case, the result was <u>one</u> too <u>low</u> in the 12th digit <u>only</u>.

References

JPC 25 (page 50) first version by Laurent Istria.

JPC 41 (page 32) second version by Guy Toublanc.

Related Keywords

COMB, FACT

The HP-28 obtains the same wrong answer for PERM(84,83) etc.
It gets the correct answer for PERM(84,84) because it uses FACT(84).

WORKAROUND: Use OPTION ROUND POS...
but then other values fail.

ARR (continued)

Authors

Laurent Istria and Guy Toublanc.

ASC$

ASC$ (ASCII string) returns a string stripped of all non-displayable ASCII characters.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
ASC$ ( string )
```

Example

`DISP ASC$("AbC"&CHR$(27))`	Displays the string "AbC.". The period takes the place of the Escape character (27).

Input Parameter

Item	Description	Restrictions
string	String expression.	None.

Operation

The ASCII character set :

The ASCII (American Standard Code for Information Interchange) code is a character set widely used by computers.

In this standard, the numerical value of displayable characters are in the range from 32 to 126. Values between 0 and 31 as well as 127 are used to control data transmission and can not be displayed. Finally, characters above 128 are undefined in the standard ASCII character set.

The ASC$ function :

ASC$ returns its input argument, with all non displayable characters replaced by a period (".").

References

JPC 22 (page 31) first version by Michel Martinet.

JPC 27 (page 34) second version by Michel Martinet.

Related Keywords

ATH$, HTA$

Authors

Pierre David and Michel Martinet.

ATH$

ATH$ (Ascii To Hexadecimal) returns the hexadecimal string corresponding to the parameter string.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
ATH$ ( string )
ATH$ ( string , mode )
```

Examples

`A$=ATH$("ABCDE")`	Stores the hexadecimal equivalent "1424344454" in A$.
`A$=ATH$("ABCDE",1)`	Stores the standard hexadecimal equivalent "4142434445" in A$.

Input Parameters

Item	Description	Restrictions
string	String expression.	None.
mode	Numeric expression. Default : 0	None.

Operation

ATH$ returns a string of hexadecimal digits corresponding to its argument.

This hexadecimal string can have two different formats according to *mode* :

If *mode* = 0, logical value *false* (default), the order of the two hexadecimal digits that represent an ASCII character is reversed. For example, character "A" (hexadecimal code 41) will be translated into "14". This representation is similar to the internal data format in the HP-71.

If *mode* is different from 0, logical value *true*, a standard representation is used. Character "A" (hexadecimal code 41) will be translated into "41".

References

JPC 22 (page 31) first version by Michel Martinet.

JPC 27 (page 34) second version by Michel Martinet.

ATH$ (continued)

To be published : third version by Pierre David.

Related Keywords

`HTA$`, `ASC$`

Authors

Pierre David and Michel Martinet.

ATTN

ATTN (ATTeNtion) enables or disables the action of the [ATTN] key to stop program execution.

■ Statement	■ Keyboard Execution
O Function	O CALC Mode
O Operator	■ IF...THEN...ELSE
	■ Device Operation

```
ATTN ON
ATTN OFF
```

Examples

```
10 ATTN OFF
20 REPEAT
30   K$=KEYWAIT$
40   DISP K$
50 UNTIL K$="#43"
60 ATTN ON
```

Defines a loop to display all keys pressed until the user presses [ATTN].

```
ATTN OFF @ BEEP INF,INF
```

Don't try this example ! The only way to stop it is INIT 1.

Operation

The [ATTN] key :

Generally, the [ATTN] key stops program execution. You have to press [ATTN] twice to stop the execution of some functions found in the HP-IL, Math or JPC Rom modules.

The ATTN command :

ATTN OFF deactivates the action of the [ATTN] key. This means that you will not be able to stop program or function execution with the [ATTN] key. While you are in this mode, pressing [ATTN] loads keycode "#43" into the key buffer and the keycode is processed as any other standard keycode.

However, during data or command input, the [ATTN] key clears the input line even if ATTN OFF has been executed. ATTN OFF only inhibits program break with the [ATTN] key.

Caution : the only way to stop a program while in the ATTN OFF mode is to execute a level one initialization INIT 1. This also restores the main environment and variables.

ATTN OFF disables the action of [ATTN], however this has no effect on INPUT or LINPUT. To mask the effect of this key, it must be redefined to a null string. This is done as follows :

```
10 DEF KEY "#43","";
20 INPUT A$
```

and setting the HP-71 to USER mode. Then, the [ATTN] key has no effect.

An other way is to use the statement FINPUT.

ATTN (continued)

ATTN ON re-activates the normal operation of the [ATTN] key.

References

JPC 23 (page 38) by Pierre David and Michel Martinet.

Related Keywords

USER, DEF KEY, FINPUT

Authors

Pierre David and Michel Martinet.

BELL

BELL causes the printer's beeper to sound if possible.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	○ Device Operation

```
BELL
```

Example

`IF DEVADDR("HP82905B")>0 THEN BELL`	If there is an HP82905B printer on the loop, then it will beep.

Operation

BELL causes the peripheral specified by the last PRINTER IS command to beep, if it is able to do so.

The ThinkJet has no beeper.

Codes sent to the printer :

Character code 7.

References

JPC 26 (page 39) first version by Pierre David.

JPC 40 (page 16) second version by Pierre David.

Also consult your printer reference manual.

Related Keywords

BOLD, MODE, PAGELEN, PCR, PFF, PLF, UNDERLINE, WRAP

Author

Pierre David

BOLD

BOLD enables or disables the bold mode of the printer.

■ Statement	■ Keyboard Execution
O Function	O CALC Mode
O Operator	■ IF...THEN...ELSE
	O Device Operation

```
BOLD ON
BOLD OFF
```

Examples

`BOLD ON @ PRINT "JPC"`	Enables bold print and prints "JPC".
`BOLD OFF @ PRINT "JPC"`	Disables bold print and prints "JPC".

Operation

BOLD ON enables bold print on the peripheral designated by PRINTER IS. BOLD OFF returns to normal print. The action of this statement depends on the peripheral used. It is intended for peripherals using the Hewlett-Packard Printer Command Language (*PCL*).

Codes sent to the printer :

```
BOLD ON  :ESC ( s 1 B
BOLD OFF:ESC ( s 0 B
```

References

JPC 26 (page 39) first version by Pierre David.

JPC 40 (page 16) second version by Pierre David.

Also consult your printer reference manual.

Related Keywords

BELL, MODE, PAGELEN, PCR, PFF, PLF, PRINT, PRINTER IS, UNDERLINE, WRAP

Author

Pierre David

CASE

CASE is part of SELECT ... CASE ... END SELECT structure.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
CASE element , ...
CASE relational operator  element , ...
CASE element TO element , ...
CASE ELSE
```

Examples

`CASE 8,5 TO 7,<0,>=10`	Selects this case if expression in SELECT is equal to 8, if it is between 5 and 7 or negative or greater or equal to 10.
`CASE >"Z","A" TO "BCD","0" TO "9"`	Selects this case if expression in SELECT is greater than "Z" or is between "A" and "BCD" or between "0" and "9".

Input Parameters

Item	Description	Restrictions
element	Numeric or alphanumeric expression.	All expressions must have the same type.
relational operator	<, =, >, <=, >=, <>, # and ?	None.

Operation

CASE is one of the components of the choice structure SELECT ... END SELECT.

CASE offers a choice of expressions. If the selected expression matches a CASE choice, execution will resume at the statement following the selected CASE.

References

JPC 52 : first version by Pierre David and Janick Taillandier.

HP 9000 series 200/300 Basic 4.0

CASE (continued)

Related Keywords

`SELECT ... END SELECT`

Authors

Pierre David and Janick Taillandier.

CENTER$

CENTER$ adds spaces at the beginning of the string specified in parameter in order to center it.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
CENTER$ ( string , width )
```

Example

`A$=CENTER$("Centered string", 22)`	Stores into A$ 3 spaces followed by the string specified in parameter.

Input Parameters

Item	Description	Restrictions
string	Alphanumeric expression.	None.
width	Numeric expression rounded to an integer.	1 through 524287.

Operation

CENTER$ adds spaces before the string specified, so that this string is at the center of a *width* characters string.

Leading and lagging spaces are first removed from the parameter string (see ~~REDUCE$~~ RED$).

BUT NO! CENTER$("   HI!", 22) IS NOT CENTERED.*

References

JPC 21 (page 34) first version of the Basic text formatter by Pierre David.

JPC 26 (page 50) second version of the Basic text formatter with assembly language functions by Pierre David and Michel Martinet.

*FIXED IN REV X.

Related Keywords

CESURE, FORMAT$, RED$, REDUCE$, SPACE$

Authors

Pierre David and Michel Martinet.

CESURE

CESURE returns the position of the first place in the string where a word-break can occur.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
CESURE ( string , width )
```

Example

```
A=CESURE("PPC Paris",7)
```

Stores 3 in variable A : word break can occur at the third character.

Input Parameters

Item	Description	Restrictions
string	Alphanumeric expression.	None.
width	Numeric expression rounded to an integer.	1 through 524287.

Operation

CESURE scans the string from the character specified by *width* back to the beginning of the string, looking for a place where a word-break can occur.

CESURE handles standard punctuation marks : *question mark*, *exclamation mark*, *semicolon*, *colon*, *period* and *opening bracket*. The algorithm is devised so that the string will not be cut in front of one of these marks.

References

JPC 21 (page 34) first version of the Basic text formatter by Pierre David.

JPC 26 (page 50) second version of the Basic text formatter with assembly language functions by Pierre David and Michel Martinet.

Related Keywords

CENTER$, FORMAT$, RED$, REDUCE$, SPACE$

CESURE (continued)

Authors

Pierre David and Michel Martinet.

COMB

COMB (combinations) computes the number of possible different sets of n items taken p at a time.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
COMB ( n , p )
```

Example

`A=COMB(4,3)` Stores 4 into variable A.

Input Parameters

Item	Description	Restrictions
n	Numeric expression.	Integer between 0 and 10^{12}-1.
p	Numeric expression.	Integer between 0 and n.

Operation

COMB(n,p) = C_n^p = n! / (p! * (n-p)!)

COMB computes the number of possible different sets (combinations) of n items taken p at a time, not counting re-arrangements.

In order to increase the range of valid parameters, and to improve accuracy, ARR uses a multiplication based algorithm instead of factorials. This results in long execution times for large numbers.

References

JPC 25 (page 50) first version by Laurent Istria.

JPC 41 (page 32) second version by Guy Toublanc.

Related Keywords

ARR, FACT

Authors

Laurent Istria and Guy Toublanc.

CONTRAST

CONTRAST returns the current contrast setting.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
CONTRAST
```

Example

```
A=CONTRAST
```

Operation

CONTRAST without a parameter returns the current contrast setting. This value can be changed by the statement CONTRAST followed by an expression whose value falls between 0 and 15.

References

JPC 22 (page 42) first version by Laurent Istria.

JPC 24 (page 41) second version by Jean-Jacques Moreau.

Forth / Assembler Owner's Manual (page 52). A sample Forth primitive returning the current contrast setting.

Related Keywords

CONTRAST

Authors

Laurent Istria and Jean-Jacques Moreau.

DATEADD

DATEADD (DATE ADDition) computes the date corresponding to the specified date incremented by the specified number of days.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
DATEADD ( date , days )
```

Examples

`A=DATEADD (7.041776, 73048)`	Stores 7.041976 (July 4th, 1976) in variable A, in DMY mode (default mode).
`DATEADD (DATE$, -1)`	Returns yesterday date.
`A=DATEADD (1.011986, 364)`	Stores 31.121986 (December 31, 1986) in variable A, in DMY mode.

Input Parameters

Item	Description	Restrictions
date	Numeric expression interpreted according to current format, or alphanumeric expression.	From October 15, 1582 through December 31, 9999.
days	Numeric expression rounded to an integer.	negative or positive.

Operation

DATEADD computes the date corresponding to the specified date incremented by the specified number of days.

For a complete description of date formats see DATESTR$.

References

JPC 28 (page 40) first version by Laurent Istria.

JPC 28 (page 35) second version by François Le Grand.

JPC 49 (page 24) third version by Pierre David et Janick Taillandier.

HP-41 Time Module Owner's Manual.

DATEADD (continued)

The keyword for DATEADD was previously DATE+.

Related Keywords

DATE$, DATESTR$, DDAYS, DMY, MDY

Authors

Pierre David, Laurent Istria, François Le Grand and Janick Taillandier.

DATESTR$

DATESTR$ (DATE to STRing) converts a date to the HP-71 string format for date : "yyyy/mm/dd".

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
DATESTR$ ( date )
```

Example

`A$=DATESTR$ (14.071789)`	Stores "1789/07/14" in variable A$. IF DMY MODE IS ACTIVE

Input Parameter

Item	Description	Restrictions
date	Numeric expression interpreted according to current format, or alphanumeric expression.	From October 15, 1582 through December 31, 9999.

Operation

Date formats :

The basic HP-71 supports two date formats :

String format :

Dates expressed in this format are alphanumeric strings with two or four digits for the year, two digits for the month and two digits for the day. They can be represented by : *yyyy/mm/dd* or *yy/mm/dd*.

For example, 1987/05/15 or 87/05/15 are valid date specification (May 15, 1987).

If the year is coded on two digits, it will be interpreted as 19*yy* if *yy*> =60, or as 20*yy* if *yy*<60.

Date functions in JPC Rom support both string formats.

Standard numeric format :

Dates are expressed as a number of the form : *yyddd*, where *yy* represents the year and *ddd* the day in year.

For example, May 15, 1987 is represented by 87135. Year is 1987 and May 15 is the 135th day in this year.

This format is hard to use. It is primarily used, on the "basic" HP-71, for date computations.

DATESTR$ (continued)

JPC Rom provides a more convenient alternative that uses the same format as the HP-41 Time Module.

JPC Rom numeric format :

This format allows date input using European or American format. You can choose either mode with the DMY and MDY keywords.

In DMY (Day Month Year) mode, during inputs, dates are interpreted as *dd.mmyyyy*. So, May 15, 1987 is represented as : 15.051987.

In MDY (Month Day Year) mode, dates are interpreted as *mm.ddyyyy*. So, May 15, 1987 is represented as : 5.151987.

The choice between both modes is reflected by the system flag -53. This flag is clear in MDY mode (default mode) and set in DMY mode.

Supported formats :

Date functions in JPC Rom support two date formats :

- dates in string format (*"yyyy/mm/dd"* or *"yy/mm/dd"*), or

- dates in numeric format (*dd.mmyyyy* in DMY mode, or *mm.ddyyyy* in MDY mode).

The DATESTR$ function :

DATESTR$ converts a date from JPC Rom numeric format (*dd.mmyyyy* or *mm.dddyyy*) to string format (*"yyyy/mm/dd"*).

It can be used with other date functions to easily isolate a date component.

References

JPC 49 (page 24) third version of DATELEX including DATESTR$ by Pierre David and Janick Taillandier.

Related Keywords

DATE$, DMY, MDY, SETDATE

Authors

Pierre David and Janick Taillandier.

DBLIST

DBLIST (Display Basic LIST) produces a structured listing of a Basic program.

INDENT
E1/6D

■ Statement	■ Keyboard Execution
O Function	O CALC Mode
O Operator	■ IF...THEN...ELSE
	■ Device Operation

```
DBLIST [ INDENT indentation ][TO target ]
DBLIST file [ INDENT indentation ][TO target ]
DBLIST file , start line [ INDENT indentation ][TO target ]
DBLIST file , start line , final line [ INDENT indentation ][TO target ]
```

Examples

`DBLIST MYSUB INDENT 3`	List program MYSUB, from the first to the last line, indenting structures by 3 spaces.
`DBLIST MYSUB,10`	List line 10 of program MYSUB, without structure indenting.
`DBLIST MYSUB,10,100 INDENT 2 TO LISTE`	List program MYSUB, from line 10 to line 100, indenting structures by 2 spaces. The output is sent to file LISTE.

Input Parameters

Item	Description	Restrictions
file	String expression or unquoted string. Default : current file.	File name with optional device specifier.
start line	Integer constant identifying a program line. Default : First program line.	1 through 9999.
end line	Integer constant identifying a program line. Default : Start line, if specified ; otherwise, last program line in file.	Start line through 9999.
indentation	Numeric expression rounded to an integer. Default : 0 REV x DEFAULT: 2	0 through ~~255~~. 1048575
target	String expression or unquoted string. Default : Listing is output on current DISPLAY IS device.	File name with optional device specifier.

Operation

DBLIST produce a "structured" listing of a Basic file on the current DISPLAY IS device or on the LCD display if no device has been specified.

Basic line numbers are justified, with a space to the left, to be 4 characters long. So all lines are aligned, no matter their line number.

DBLIST (continued)

DBLIST does not output line numbers for lines containing only comments (beginning with !, but not with REM). A dash (-) is output to mark the first comment line in a series. The statement RENUMREM is intended to ease the renumbering of comment lines. With this special processing, comment lines are no longer considered as standard Basic lines.

DBLIST skips a line before a function definition (DEF FN), a DATA block or a label. A line is also skipped after function definitions and DATA blocs. So, the various building blocs that make-up your program are well separated.

DBLIST skips a line, draws a line and skips another line before a sub-program (SUB). This emphasizes independent program blocs.

Finaly, DBLIST allows indenting of logical structures. The body of logical structures, whether a standard (FOR...NEXT) or a JPC Rom (WHILE...END WHILE loop or tests or SELECT) is shifted to the right by the number of spaces specified in the *indentation* value of INDENT. Default value is 0, i.e. structures are not indented.

Structure indenting can help identify invalid structures in programs. This is a very useful complement to the structured programming statements provide by JPC Rom.

The TO option allows you to direct the output to a text file of your choice. Incidentally, this option is used to prepare HP-71 Basic program listings published in the Journal of *PPC Paris*. The file is created, then filled. If the file already exists, the error JPC ERR:File Exists is generated.

The current DELAY setting determines how long each line will be displayed. The WIDTH setting determines the width of the displayed line.

To halt a listing and display the cursor, simply press [ATTN].

References

JPC 18 (page 25) first version of Basic program JPCLISTE by Pierre David and Michel Martinet.

JPC 38 (page 24) first version of BLIST by Jean-Pierre Bondu.

To be published : second version of DBLIST by Pierre David and Janick Taillandier.

DBLIST was called BLIST.

Related Keywords

DELAY, LIST, PBLIST, PLIST, WIDTH, all structured programming keywords

Authors

Jean-Pierre Bondu, Pierre David and Janick Taillandier.

DDAYS

DDAYS (Delta DAYS) compute the number of days between two dates.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
DDAYS ( date1 , date2 )
```

Examples

BASTILLE DAY INDEPENDENCE DAY

`A=DDAYS(7.141789,7.041776)`	Stores 4758 days between July 14th, 1789 and July 4th, 1776, using MDY mode.
`A=DDAYS(1.011986,31.121986)`	Stores 364 days in variable A, using DMY mode.
`DISP DDAYS(DATE$,7.041776)`	Computes and displays the number of days between July 4th, 1776 and today.

Input Parameters

Item	Description	Restrictions
date1, date2	Numeric expressions interpreted according to current format, or alphanumeric expressions.	From October 15, 1582 through December 31, 9999.

Operation

DDAYS compute the number of days between *date1* and *date2*. If *date1* comes after *date2*, the result will be positive.

For a complete description of date formats, see DATESTR$.

References

JPC 28 (page 40) first version by Laurent Istria.

JPC 49 (page 24) third version by Pierre David et Janick Taillandier.

HP-41 Time Module Owner's Manual.

Related Keywords

DATEADD, DATESTR$, DMY, MDY

Authors

Pierre David, Laurent Istria and Janick Taillandier.

DDIR

E1/6E

DDIR (Display DIRectory) lists directory of the specified device.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	○ Device Operation

```
DDIR [ TO target ]
DDIR file specifier [ TO target ]
DDIR ALL [ TO target ]
```

Examples

`DDIR :TAPE`	Lists directory of mass storage unit `:TAPE`.
`DDIR :PORT(0) TO LISTE`	List directory of port number 0 into file LISTE.
`DDIR ALL`	Lists all files in HP-71.
`DDIR ESSAI:TAPE(2) TO A$`	Lists all files after file ESSAI on mass storage unit `:TAP(2)` into the file whose name is specified by A$.

Input Parameters

Item	Description	Restrictions
file specifier	String expression or unquoted string. Default : `:MAIN`	Device specifier or file specifier with optional device specifier.
target	String expression or unquoted string. Default : Listing on `DISPLAY IS` device.	File specifier with optional device specifier.

Operation

The keyword DDIR lists the directory of the specified unit on the display device. The list is similar to the one produced by the standard keyword CAT.

The current DELAY setting determines how long the HP-71 displays each line. We recommand you to use a DELAY *x*, 8 which eases the LCD display reading.

File specification

The DDIR syntax allows to choose a peripheral or a part of a directory.

DDIR (continued)

Device specifier only

If you provide a the device name alone, only the directory of this unit will be listed. For example :

- DDIR :PORT(0.01) lists the directory of the port number 0.01,
- DDIR :TAPE lists the directory of the HP-IL mass storage device,
- DDIR :PORT lists the contents of *all* HP-71 ports,
- DDIR :MAIN lists only the contents of main memory.

Both file and device specifiers

If you specify both a file and a device, the listing will begin starting from this file until the last file in the device. For example :

- DDIR ESSAI:MAIN lists the directory of main memory after file ESSAI included,
- DDIR ESSAI:TAPE lists the directory of mass storage unit after file ESSAI included,
- DDIR ESSAI:PORT(0.01) lists the directory of port number 0.01 after file ESSAI included,
- DDIR ESSAI:PORT looks for the file in all ports, and lists the directory of the founded port.

Special cases

DDIR ALL lists the directory of all files in the HP-71.

DDIR lists only the directory of the main memory.

DDIR followed by a file specifier, without a device specifier, looks for the file in all memory, then lists the remaining files in the corresponding unit (port or main memory).

Output redirection

When DDIR is followed by TO, then by a file specifier, the listing is stored into this file. Nothing is displayed. When output redirection is required, DDIR and PDIR are equivalent.

If the file already exists, the error ERR:File Exists is reported.

Data stored into this file share the same format as those resulting from CAT$. Please, refer to this function for more details.

This feature is similar to the one provided by statements PDIR, DBLIST and PBLIST.

Example of use

Redirection is useful because it allows you to execute an action on all files of a given device. For example :

```
100 DIM P$[8],F$[43],T$[8],A
110 T$="TMP"                  ! temporary file
120 FINPUT P$,"Device: :MAIN","8PU",A
130 IF NOT A THEN END
140 SFLAG -1 @ PURGE T$ @ CFLAG -1
150 DDIR ":"&P$ TO T$
160 ASSIGN #1 TO T$
170 LOOP
180   READ #1;F$
190   F$[POS(F$," ")]=""      ! removes unused characters
  -   what you want to do
200   SECURE F$&":"&P$        ! for example...
  -   done...
210 END LOOP
```

This simple example execute an action on line 200 for all files in the specified device. By changing this action, you can easily copy files from one port to another, purge files, rename them, etc.

References

To be published : first version by Jean-Jacques Dhénin.

Related Keywords

CAT$, CAT, DBLIST, PBLIST, PDIR

Author

Jean-Jacques Dhénin

DMY

DMY (Day Month Year) enable date input in numeric format *dd.mmyyyy*.

■ Statement	■ Keyboard Execution
O Function	O CALC Mode
O Operator	■ IF...THEN...ELSE
	■ Device Operation

```
DMY
```

Example

DMY

Operation

In the mode selected by DMY, numeric date parameters used by JPC Rom date functions must use the *dd.mmyyyy* numeric format.

Keep in mind that string format is independent of the DMY / MDY modes. It can always be used. For example, to compute the day corresponding to July 4, 1789, you can use either of the following expressions in DMY mode :

DOW$(4.071776) or DOW$("1776/07/04")

For a complete description of date formats, see DATESTR$.

STORED IN FLAG -53

FLAG(-53) = 0 → MDY
= 1 → DMY

References

JPC 28 (page 40) first version by Laurent Istria.

JPC 49 (page 24) third version by Pierre David et Janick Taillandier.

HP-41 Time Module Owner's Manual.

Related Keywords

DATEADD, DATESTR$, DDAYS, DOW, DOW$, MDY

Authors

Pierre David, Laurent Istria and Janick Taillandier.

DOW

DOW (Day Of Week) returns the day of week corresponding to the specified date parameter.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
DOW
DOW ( date )
```

Examples

`A=DOW(1.011986)`	Stores in A the day number in the week corresponding to January 1, 1986.
`A=DOW(DATE$)`	Returns the day number corresponding to today. This give the same result as `DOW` alone.
`DISP DOW`	Display day number for today.

Input Parameter

Item	Description	Restrictions
date	Numeric expression interpreted according to current format, or alphanumeric expression. Default : today	From October 15, 1582 through December 31, 9999.

Operation

DOW return the day of week for a given date as a number. So, you can easily use this value in your programs. For a complete description of date formats, see DATESTR$.

For example, to display French day names :

DOW (continued)

```
100 SELECT DOW
110   CASE 0
120     A$="Dimanche"
130   CASE 1
140     A$="Lundi"
150   CASE 2
160     A$="Mardi"
170   CASE 3
180     A$="Mercredi"
190   CASE 4
200     A$="Jeudi"
210   CASE 5
220     A$="Vendredi"
230   CASE 6
240     A$="Samedi"
250 END SELECT
260 DISP DATE$;" : ";A$
```

0 corresponds to Sunday, 1 to Monday... and 6 to Saturday.

References

JPC 17 (page 25) day of week computation in Basic by Pierre David.

JPC 28 (page 40) first version by Laurent Istria.

JPC 49 (page 24) third version by Pierre David et Janick Taillandier.

HP-41 Time Module Owner's Manual.

Related Keywords

DATE$, DATESTR$, DMY, DOW$, MDY, SETDATE

Authors

Pierre David, Laurent Istria and Janick Taillandier.

DOW$

DOW$ (Day Of Week) returns the name of the day corresponding to the specified date or today.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
DOW$
DOW$ ( date )
```

Examples

`A$=DOW$(1.011986)`	Stores the string Wednesday in variable A$.
`DISP DOW$`	Display current day name.

Input Parameter

Item	Description	Restrictions
date	Numeric expression interpreted according to current format, or alphanumeric expression. Default : today	From October 15, 1582 through December 31, 9999.

Operation

DOW$ returns the day corresponding to the specified date.

If no date is specified, DOW$ returns the day corresponding to current date.

Day names are expressed in English. These names correspond to messages included in JPC Rom. You can used the function MSG$ (in Forth/Assembler module or Text Editor or available through the User's Library) to get all days in a week.

Sunday corresponds to message 225008+0, Monday to message 225008+1, and so on to Saturday message 225008+6.

As day names are stored in a message table, it is possible to use a translator Lex to translate the names.

References

JPC 17 (page 25) day of week computation in Basic by Pierre David.

DOW$ (continued)

JPC 28 (page 40) first version by Laurent Istria.

JPC 49 (page 24) third version by Pierre David et Janick Taillandier.

HP-41 Time Module Owner's Manual.

Related Keywords

`DATE$`, `DATESTR$`, `DMY`, `DOW`, `MDY`, `SETDATE`

Authors

Pierre David, Laurent Istria and Janick Taillandier.

EDIT allows merging of Lex files, or editing files on external peripherals. EDIT is nonprogrammable.

■ Statement	■ Keyboard Execution
O Function	O CALC Mode
O Operator	O IF...THEN...ELSE
	O Device Operation

```
EDIT
EDIT file1
EDIT [file1] TO file2        REV X: file1 is optional.
```

Examples

`EDIT AREUH:TAPE`	File AREUH is copied from :TAPE to main memory, and becomes current file.
`EDIT AREUH:TAPE TO TOTO:PORT(0)`	File AREUH is copied from :TAPE to :PORT(0), changes its name and becomes current file.
`EDIT STRINGLEX`	Edit Lex file STRINGLEX.

Input Parameters

Item	Description	Restrictions
file1	String expression or unquoted string. Default : System workfile	File name with optional external device specifier.
file2	String expression or unquoted string. Default : File with same name in main Ram.	The device specifier must be in Ram.

Operation

Copying and editing files :

If the first file specifier indicates an external mass memory device, the file is first copied into the HP-71.

If a second file specifier is provided, the first file is copied into it. Then the file is made the current workfile.

So EDIT on external files is similar to COPY followed by a standard EDIT on this file.

If the type of the copied file is invalid (i.e. different from Basic, Keys or Lex), the copy is done and `ERR:Invalid Filetype` is reported.

Chaining Lex files :

EDIT (continued)

The edited file can be a Lex file. This is the first step in linking Lex files. See MERGE for further details.

Caution !

When you edit a Lex file, it becomes the current workfile. If you execute a PURGE command on this file, the workfile is not changed to the standard `workfile`, this yields to strange results.

To prevent this be sure to do an `EDIT` to edit the system workfile after you finish merging (linking) Lex files.

References

JPC 31 (page 54) editing files on external peripherals by Jean-Pierre Bondu.

JPC 23 (page 47) Basic program to merge Lex files by Michel Martinet.

JPC 37 (page 22) assembly language merging of Lex files by Pierre David and Michel Martinet.

Related Keywords

COPY, EDIT, MERGE

Authors

Jean-Pierre Bondu, Pierre David and Michel Martinet.

ENDUP

ENDUP defines a command string to be executed when the HP-71 turns off.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

ENDUP *command string*

Example

`ENDUP "BEEP@'Bye...'"`	The HP-71 will beep and display "Bye..." each time it turns off.

Input Parameter

Item	Description	Restrictions
command string	String expression.	0 through 95 characters.

Operation

The ENDUP command string can include any instruction you wish, provided that it can be executed from the keyboard.

When you execute ENDUP, the command string is stored without checking for syntactical errors. The computer may have only one ENDUP string at any given time. When you turn the HP-71 off, the ENDUP string is executed if it is error free. Otherwise, an error is reported and the computer is left in a state such that you have only to push on [ATTN] to turn it off.

The specified string is kept in a buffer. See ADBUF$ for more informations on buffers and their use.

Note : the string specified by ENDUP is not executed when the HP-71 is turned off in CALC mode or within KA.

References

JPC 25 (page 43) first version by Jean-Jacques Moreau.

JPC 31 (page 29) second version by Jean-Jacques Moreau.

Related Keywords

ADBUF$, ENDUP$, STARTUP$, STARTUP

Author

Jean-Jacques Moreau

ENDUP$

ENDUP$ returns the command string specified in ENDUP.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
ENDUP$
```

Example

A$=ENDUP$	Stores into A$ the command string to be executed when the HP-71 is powered off.

Operation

ENDUP$ returns the command string to be executed when the HP-71 is powered off. The length of this string cannot be greater than 95 characters.

If no command has been specified by ENDUP, ENDUP$ returns a null string.

References

JPC 25 (page 43) first version by Jean-Jacques Moreau.

JPC 31 (page 29) second version by Jean-Jacques Moreau.

Related Keywords

ENDUP, STARTUP, STARTUP$

Author

Jean-Jacques Moreau

ENTRY$

ENTRY$ (entry point) returns the entry point address for the specified keyword.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
ENTRY$ ( keyword )
ENTRY$ ( keyword , sequence )
```

Examples

`A$=ENTRY$("ENTRY$")`	Stores execution address of ENTRY$ in variable A$.
`DISP ENTRY$("EDIT",2)`	Returns the address of the second EDIT, i.e. system EDIT. Lex files are searched before standard functions.

Input Parameters

Item	Description	Restrictions
keyword	String expression.	The keyword must exist.
sequence	Numeric expression rounded to an integer. Default : 1	The keyword must exist.

Operation

ENTRY$ returns the entry point address of the specified function or statement. This address is equivalent to the start address of the run-time execution code.

ENTRY$ is specially useful when used with the Debugger (HP-82178A) to easily locate entry points.

Caution : files in HP-71 main memory are frequently moved. For example, if a file is purged or its size changed and if it is located before the Lex file containing the function, the entry point address will change. You can avoid these problems by keeping code under study in independent Ram.

The entry point is the address of the execution code, or the address specified by the ENTRY pseudo-op used by the HP-71 Forth / Assembler Rom.

If a second parameter is provided, ENTRY$ looks for the function in all available Lex files. This includes all functions provided by the built-in operating system.

If the keyword does not exist or if the sequence number is greater than the number of times the keyword occurs in your HP-71, ENTRY$ will return the ERR:`Invalid Arg` error.

ENTRY$ (continued)

For keywords of more than 8 characters in length, special processing is required from the system. So, keywords like UNDERLINE or RANDOMIZE are recognized as UNDERLIN or RANDOMIZ. The final "E" is processed by the function itself. ENTRY$ cannot process these extra characters. ENTRY recognize UNDERLIN and don't take care of the "E". So, ENTRY$("RANDOMIZE") and ENTRY$("RANDOMIZ----") ignore extra characters and return the same address.

The keyword found is the longest keyword corresponding to the characters specified, others are ignored. So, ENTRY$("MEMORY") returns the entry point address of function MEM.

References

JPC 31 (page 22) first version by Jean-Jacques Moreau.

Forth/Assembler Rom Owner's Manual (page 63).

Internal Design Specification, Volume I.

Related Keywords

ADDR$, LEX, PEEK$, TOKEN, SYSEDIT

Author

Jean-Jacques Moreau

ESC$

ESC$ (ESCape) returns the string with a leading "escape" character.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
ESC$
ESC$ ( string )
```

Examples

`PRINT ESC$("Y")`	Puts a printer such as the ThinkJet into monitor mode : all characters received will be printed.
`PRINT ESC$("*b80W")&G$`	Sends a graphic line to a ThinkJet or LaserJet.
`DISP ESC$&"j";`	Enables the Roman8 character set on an HP92198B video interface.

Input Parameter

Item	Description	Restrictions
string	Alphanumeric expression. Default : Null string.	None.

Operation

Escape sequences :

Escape sequences are used by most computers to control peripherals. For example, the HP-71 uses escape sequences to control the internal LCD display and with HP-IL peripherals.

An escape sequence is prefixed by a character "escape" or *ESC* (code 27). It is recognized by the peripheral as the beginning of a command and not as normal data.

The *ESC* code is followed by a string coding the command. If the peripheral recognizes it, it will respond accordingly.

For example, with a ThinkJet, if you execute :
`PRINT "THE HP-71";`
the printer will print "THE HP-71". Now, if you try :
`PRINT CHR$(27)&"&dD";`

the printer will interpret the 4 characters as a command to enter underline mode, the characters will not be printed. This is an escape sequence.

You don't have to remember the most frequently used escape sequences for the ThinkJet. You can use statements like `BOLD`, `PAGELEN`, `UNDERLINE` or `WRAP`.

The ESC$ function :

The `ESC$` function adds an *escape* character before the specified string. If no string is specified, `ESC$` is equivalent to `CHR$(27)`.

References

JPC 26 (page 39) first version by Pierre David.

JPC 40 (page 16) second version by Pierre David.

Consult the reference manuals of your peripherals...

Related Keywords

`BOLD`, `CHR$`, `PAGELEN`, `UNDERLINE`, `WRAP`

Author

Pierre David

EXECUTE

EXECUTE executes the specified command string and stops program execution.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	○ IF...THEN...ELSE
	■ Device Operation

```
EXECUTE command string
```

Example

```
10 EXECUTE "FREEPORT(0)@RUN,'A'"
20 'A':
```

Switches port 0 to independent Ram and resumes execution at label 'A'.

Input Parameter

Item	Description	Restrictions
command string	Alphanumeric expression.	0 through 95 characters.

Operation

EXECUTE executes the *command string* and stops program execution.

This allows "programming" of some non-programmable functions.

EXECUTE should never be used in a subprogram or loop and choice structures such as LOOP, IF or SELECT : it destroys calling environments.

The programme is considered as *executing* until the whole string has been executed. This allows using CONT to resume programme execution.

References

JPC 31 (page 29) second version of ENDUPLEX by Jean-Jacques Moreau.

Related Keywords

ENDUP, STARTUP

Author

Jean-Jacques Moreau

EXIT

EXIT exit a FOR ... NEXT loop.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
EXIT loop variable
```

Example

```
10 FOR I=1 TO INF
20   IF FNC(I) THEN EXIT I
30 NEXT I @ BEEP
```

Exits the FOR ... NEXT loop and resumes execution at the instruction that follows NEXT I (BEEP) if FNC(I) is different from 0.

Input Parameter

Item	Description	Restrictions
loop variable	Simple numeric variable.	None.

Operation

EXIT exits conveniently from a FOR ... NEXT loop. Informations necessary to control the loop are cleared.

Normal loop exit is through statement NEXT when the loop counter exceeds the final value specified.

On some occasions it is useful to exit a loop prematurely, whenever special conditions are met. EXIT provides an elegant solution for handling such situations. For example the following programs compute 10 squared roots, unless an argument is negative :

```
10 DATA 1,2,3,4,5,-6,7,8,9,10
20 FOR I=1 TO 10
30   READ X
40   IF X<0 THEN EXIT I
50   DISP SQRT(X)
60 NEXT I
70 DISP 'Ended'
```

References

JPC 30 (page 49) first version by Janick Taillandier.

Related Keywords

`FOR` ... `NEXT`, `LEAVE`

Author

Janick Taillandier

FILESIZE

FILESIZE returns the size of the specified keyword.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	○ Device Operation

```
FILESIZE ( file )
```

Example

```
A=FILESIZE("ESSAI")
```

Returns the size of file ESSAI if found, 0 otherwise.

Input Parameter

Item	Description	Restrictions
file	String expression.	Filename with optional device specifier.

Operation

FILESIZE returns the file size in bytes, or 0 if the file cannot be found in memory or on the specified mass media. (HALF BYTES ARE ROUNDED UP; IF FILESIZE = 99.5 BYTES, 100 IS RETURNED)

This allows easily testing if a file exists, whether it is in Ram or on an external device. We have to write something like :

```
1000 IF FILESIZE(F$&":TAPE") THEN
1010   COPY :TAPE TO F$
1020 END IF
```

The size returned is the *total* file size. It includes the file header size. This header contains the file name, type, creation date and time as well as other informations used by the system. So, this size is different from the size returned by CAT or CAT$.

It is interesting to use this size because it coresponds to the available room as returned by MEM. To copy a file from mass storage to an Independent Ram, you have only to write something like :

```
IF MEM(0)>=FILESIZE("TOTO:TAPE") THEN COPY ...
```

References

JPC 23 (page 36) keyword FILE?

FILESIZE (continued)

To be published : `FILESIZE` by Henri Kudelski.

Related Keywords

`ADDR$`, `CAT`

Author

Henri Kudelski

FIND

FIND finds a character string in a Basic program. FIND is nonprogrammable.

■ Statement	■ Keyboard Execution
O Function	O CALC Mode
O Operator	O IF...THEN...ELSE
	■ Device Operation

```
FIND string
```

Example

`FIND "OSUB 1210"`	Looks for the first occurrence of string "OSUB 1210" after the current line and sets the cursor to that line.

Input Parameter

Item	Description	Restrictions
string	String expression.	None.

Operation

FIND looks for a string in the current Basic file, after the current line.

If FIND finds the string, the line is displayed and the cursor is moved to the first character of the found string.

If the string cannot be found, the error : `JPC ERR:Not Found` is reported.

The first line of a program is not searched unless the program has just been edited with EDIT.

References

JPC 31 (page 25) first version by Jean-Jacques Moreau.

JPC 45 (page 19) second version by Janick Taillandier.

The HP-75 FETCH command.

Related Keywords

FETCH

Authors

Jean-Jacques Moreau and Janick Taillandier.

REV X: Bugs are fixed!

FINPUT

FINPUT (Formatted INPUT) creates an input mask and waits for data input from the user.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
FINPUT input , prompt , attn
FINPUT input , prompt , format , attn
```

Example

```
10 DIM I$[8]
20 FINPUT I$,"File: ",A
30 IF A=0 THEN ...
```

The user enters a filename (8 characters maximum), and FINPUT stores it in A$. If the user press [ATTN], variable A is set to 0.

Input Parameters

Item	Description	Restrictions
input	Existing string variable or array.	The variable or array must be created before you use FINPUT.
prompt	String expression or string array.	Contains only displayable characters.
format	String expression or string array. Default : STR$(LEN(prompt))&"PU"	Non null string exclusively composed of characters "U" and "P" or digits specifying a format.
attn	Numeric variable or numeric array name.	None.

Operation

Protected fields :

Briefly, protected fields may be used with INPUT or LINPUT to prevent accidental erasure of important prompts.

For example, to enter a date, the display will look like :

```
Date: Dy/Mo/Yr
```

The user has to replace only characters Dy (Day), Mo (Month) and Yr (Year) by their values. Others must not change. Here is a program to do that :

FINPUT (continued)

```
100 E$=ESC$("<")                      ! Cursor off
110 A$=ESC$(">")                      ! Cursor on
120 D$=A$&"Jr"&E$"/"&A$&"Mo"&E$&"/"&A$&"Yr"
130 DISP E$&"Date: "&D$&E$;           ! Display
140 INPUT "";I$                       ! Date input
```

First, the program is not legible in spite of the comments.

Second, the mask display is slow.

Third, if a date is entered, and the [ATTN] key is pressed, the month is cleared and cursor goes to the beginning. Press [ATTN] again nothing happens. The [ATTN] key is not enabled, you cannot stop the program. The only solution is to press [ENDLINE]. This validates the input, but that is obviously not what you wished.

Fourth, after entering the date, nothing prevents the user from keying-in additional characters. How to prevent this ?

The problem is that no character is protected to the right of the date. The HP-71 has no reason to lock the remainder of the display. So we have to display the mask and then add enough protected characters : here, 96 - 14 (length of the mask) blank characters. We add the following lines to the program :

```
121 DIM S$[82]
122 S$=""
123 S$[82]=" "
```

Then replace line 130 by :

```
129 WIDTH INF
130 DISP E$&"Date: "&D$&E$&S$;
```

When running the program, there is an unpleasant display blinking before you see the mask but, at last, you cannot enter any character past the date.

A new problem appears : press [->] after the year, the display disappears at the left of the LCD. Worst, pressing [g] [->] gives you an empty screen after some time.

Using FINPUT :

Single line FINPUT :

In its simplest form, FINPUT is an extension of the LINPUT statement that facilitates the use of protected fields.

Our Basic example can now be written :

```
100 DIM I$[6]
110 FINPUT I$,"Date: Dy/Mo/Yr","6P2UP2UP2UP",A
```

In this example, it is worth noting that :

- I$ is the target string. It must be created before using FINPUT.

- *prompt* contains what will appear on the display. All characters, protected or not, are displayed.

- the next parameter is the *format* string. Let us look at the content of this expression : 6P means that the 6 first characters are *Protected*. 2U specifies that the next 2 characters are *Unprotected*. The P indicates that the next character is protected, and so on... The final P means that the remainder of the display is protected. It is not necessary to specify 82P to finish the line.

- the last parameter, *attn*, will contain 0 if the [ATTN] key was used to exit FINPUT.

It is easy to understand that the use of protected fields is greatly simplified. FINPUT has many other features, among them :

- simplifying protection specification : describing protected fields is really easy.

- handling of the [ATTN] key : during FINPUT, the [ATTN] key, pressed once restores the default display specified by the *prompt* string. [ATTN] pressed a second time exits FINPUT and stores 0 into *attn*. The program is not interrupted and it is easy to handle the [ATTN] key using a simple test such as : `IF NOT A THEN` ...

- handling of [->] and [g] [->] : these keys no longer cause the unpleasant effect described above.

- handling of "short variables" : in the previous example, if the declaration of I$ had specified less than 6 characters, for example 3, it would not have been possible to enter more than 3 characters. FINPUT adds a new security. Programs will no longer stop with the "String Overflow" error !

FINPUT without format string :

In many occasions, you don't need such a sophisticated display management. For example, to enter a file name with INPUT, you write :

```
100 INPUT "File: ";F$
```

As a file name, in Ram, cannot have more than 8 characters, with FINPUT the program becomes :

```
100 DIM F$[8]
110 FINPUT F$,"File: ",A
120 IF NOT A THEN END
```

Now, it is impossible to enter more than 8 characters, and if the user changes its mind and presses the [ATTN] key, the program handles it simply.

The *format* string is optional. If it is not present, FINPUT uses the following defaults : STR$(LEN(*prompt*))&"PU". All characters in the *prompt* string are protected, the remainder is unprotected up to the maximum length of the result string.

Multiple line FINPUT :

The most important characteristic of FINPUT is that it can process multiple input lines. It is somewhat like a complete screen mask.

If a program needs date and time input data, it can be obtained by :

FINPUT (continued)

```
100 DIM D$[6],H$[6]
110 FINPUT D$,"Date: Dy/Mo/Yr","6P2UP2UP2UP",A
120 IF NOT A THEN END
130 ! Date processing
 :
200 FINPUT H$,"Time: Hr:Mn:Sc","6P2UP2UP2UP",A
210 IF NOT A THEN END
220 ! Time processing
```

But there is another solution :

```
100 OPTION BASE 1                  ! array will begin by 1
110 DIM I$(2)[6],M$(2),P$(2)
120 DATA Date: Dy/Mo/Yr,Time: Hr:Mn:Sc
130 DATA 6P2UP2UP2UP,6P2UP2UP2UP
160 READ M$                        ! read both prompts
170 READ P$                        ! read both format strings
180 FINPUT I$,M$,P$,A
190 IF NOT A THEN END
200 ! Date processing (I$(1))
210 ! Time processing (I$(2))
```

This last solution is more elegant than the first one when you need to input large amounts of data. All data input is done in a single operation.

Cursor keys are used to skip from one line to another. [ENDLINE] is used to validate each line.

Important notice : there are two ways to exit FINPUT and validate the input :
- pressing [RUN] which validates the current line, and
- pressing [ENDLINE] twice when the cursor is in the last line.

The *attn* variable contains the line number on which FINPUT was exited. The 0 value indicates an exit via the [ATTN] key.

Using FINPUT this way allows you to fill out an entire form in a single operation. The programmer no longer needs to be concerned with movements inside the form. FINPUT handles them !

Summary :

The variable *input* must be created before executing FINPUT.

The *prompt* string must contain only displayable characters. It may not include 0 (NULL), 27 (ESC), 13 (CR), 10 (LF) or 8 (BS) codes.

The format specification can contain letters "U" and "P" (uppercase or lowercase) preceded by an optional repetition factor to specify protected and unprotected characters. The string must not be null and the resulting format must not specify a string with more than 96 characters. So, 96P is correct, but 97P or 95P2U are not.

Simple variables are considered as arrays with only one element.

Usage :

While you enter data with FINPUT, selected keys have been assigned the functions :

[ATTN]
If characters have been keyed in, the display is restored according to *prompt*.
A second time : exit from FINPUT.

[f] [OFF]
Direct exit from FINPUT.

[ENDLINE]
Validates the current line and skips to next line. If single line, exits from FINPUT. Pressing [ENDLINE] twice on the last line and validates the input.

[RUN]
Validates the current line and exits from FINPUT. If single line, [RUN] is the same as [ENDLINE].

[^], [v], [g] [^] and [g] [v]
Change line without validation of the current line. If single line, restore the default display.

Variable contents on exit :

prompt and *format* variables are never modified.

After a normal exit (via [ENDLINE] or [RUN]), the variable *attn* contains the line number on which exit occurred. This number is between 1 and the array size.

The destination variable contains the data you entered.

When you exit by way of [ATTN] or [f] [OFF], the variable *attn* contains 0. The destination variable remains unchanged.

References

JPC 43 (page 16) FINPUT by Pierre David and Janick Taillandier.

Related Keywords

INPUT, LINPUT, DISP

Authors

Pierre David and Janick Taillandier.

FKEY

FKEY (First KEY) insert a key code at the beginning of the keyboard buffer.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

FKEY *key*

Example

```
10 DISP "Result =";R
20 K$=KEY$
30 IF NOT LEN(K$) THEN 20
40 FKEY K$
50 INPUT X$
```

Displays the result in variable R, then waits for a keystroke to go on. The key is not lost and will be used as the first character for the next data input.

Input Parameter

Item	Description	Restrictions
key	String expression.	Less than 5 characters.

Operation

FKEY adds the specified key code at the beginning of the keyboard buffer (it can hold up to 15 keystrokes). PUT adds it at the end of the buffer.

If the keyboard buffer is full the oldest keystroke is lost.

FKEY allows you to establish a priority system whereby certain instructions (key assignments) will be handled ahead of other keyboard inputs.

References

JPC 24 (page 35) first version by Jean-Pierre Bondu.

Related Keywords

KEY$, KEYWAIT$, PUT

Author

Jean-Pierre Bondu

FORMAT$

FORMAT$ inserts extra spaces inside a string so that it will have exactly the specified number of characters.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
FORMAT$ ( string , width )
```

Example

`A$=FORMAT$("J P C",9)` Adds 4 spaces inside the specified string.

Input Parameters

Item	Description	Restrictions
string	String expression.	None.
width	Numeric expression rounded to an integer.	1 through 1048575.

Operation

First, FORMAT$ reduces the string (see REDUCE$). Spaces are inserted so that the string length matches the specified length.

Spaces are inserted between words. *

This greatly facilitates the production of right and left justified formatted text.

References

JPC 21 (page 34) first version of the Basic formatter by Pierre David.

JPC 26 (page 50) second version of the Basic formatter with assembly language functions by Pierre David and Michel Martinet.

Related Keywords

CESURE, REDUCE$

* The first space to be inserted is placed after the first word; subsequent spaces are inserted between words starting from the end.

Authors

Pierre David and Michel Martinet.

FPRIM

FPRIM (First PRIMe number) returns the first prime number after the argument.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
FPRIM ( argument )
FPRIM ( argument , direction )
```

Examples

`A=FPRIM(300)`	Stores 307, first prime number after 300, into variable A.
`A=FPRIM(300,350)`	Stores 307, first prime number between 300 and 350, into variable A.
`DISP FPRIM(300,250)`	Displays 293, greatest prime number lower than 300 and higher than 250.

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	Must be an integer different from 0 and between $-10^{12}+1$ and $10^{12}-1$.
direction	Numeric expression. Default : 10^{12} * SGN(argument).	Must be an integer between $-10^{12}+1$ and $10^{12}-1$.

Operation

FPRIM returns the first prime number after the specified argument or returns the argument itself if it is a prime number.

The second parameter, *direction*, indicates whether the search must be conducted towards greater numbers (*direction* > *argument*) or lower numbers (*direction* < *argument*).

direction is also used as the upper or lower limit of the search. FPRIM returns a value of 0 if no prime is found between *argument* and *direction*.

Interrupting the function with [ATTN] :

This function can be interrupted by pressing [ATTN] twice. The HP-71 will then report the error `JPC ERR:Function Interrupted`.

References

JPC 35 (page 21) fist version of DIVILEX by Guy Toublanc.

JPC 38 (page 18) second version by Guy Toublanc.

JPC 48 (page 23) third version by Guy Toublanc.

`FPRIM` was previously called `FPRM`.

Related Keywords

`PRIM`, `NPRIM`, `DIV`

Author

Guy Toublanc

FRAC$

FRAC$ (FRACtion) approximates a real number by a fraction.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
FRAC$ ( real number )
FRAC$ ( real number , accuracy )
```

Examples

`A$=FRAC$(1.25)`	Returns the string "5/4" to variable A$.
`DISP FRAC$(PI,2)`	Display "22/7", which approximates PI at 10^{-2}.

Input Parameters

Item	Description	Restrictions
real number	Numeric expression.	None.
accuracy	Numeric expression rounded to an integer. Default : if $\lvert n \rvert >= 1$, 10, otherwise 10-exponent of n.	-99 through 99

Operation

FRAC$ gives an approximation of a real number x in the form of a fraction. The result is a character string.

The *accuracy* parameter is optional. If omitted or zero, the default accuracy is 10^{-10} if $|x| >= 1$, or $10^{-10+\text{exponent of } x}$ otherwise.

If *accuracy* is positive, precision is $10^{-accuracy}$.

If *accuracy* is negative, it represents the number of iterations to be used by the FRAC$ algorithm.

References

JPC 20 (page 55) first version of a Basic program by Pierre David.

JPC 42 (page 21) first version by Guy Toublanc.

Related Keywords

`EXPONENT`

Author

Guy Toublanc

GLINE

GLINE (Graphic LINE) builds a raster graphics representation of a drawn line for use with ThinkJet or LaserJet printers.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
GLINE x , length , first , size , gap
```

Example

```
GLINE x1,x2-x1+1,1,1,0
```
Plots a line between points x1 and x2 using variable G$.

Input Parameters

Item	Description	Restrictions
x	Numeric expression rounded to an integer.	1 through 640.
length	Numeric expression rounded to an integer.	0 through 640.
first	Numeric expression rounded to an integer.	0 through 640.
size	Numeric expression rounded to an integer.	0 through 640.
gap	Numeric expression rounded to an integer.	0 through 640.

Operation

Graphics on "raster" printers :

Printers that use raster graphics include the ThinkJet, LaserJet or QuietJet.

On such printers, graphic images are built row by row. Each line is one pixel (dot) in height. An 80 bytes area is needed to store the "image" of the 640 points in a row. GLINE and GPSET both use an 80 characters variable which must be named G$, previously created by a DIM G$[80] command.

Before storing graphic data into G$, you must initialize this variable with CHR$(0) bytes. This can be done with a code sequence like :

```
100 DIM G$[80]
110 G$=SPACE$(0,80)
```

Printing the graphic line :

Printing is done via escape sequences (see ESC$) which are device dependent. With a ThinkJet, you will execute :

```
PRINT ESC$("*r640S"); @ PWIDTH INF
```

GLINE (continued)

to initialize the graphic mode of the printer. Note this needs to be done only once for al your program.

To print each line, execute :

```
PRINT ESC$("*b80W")&G$;
```

Warning : you should execute `PWIDTH INF` before printing the graphics. This prevents the HP-71 from sending an unwanted end-of-line sequence (see `ENDLINE`) in the middle of your graphic row.

Consult your printer manual to get more informations about graphics.

Using GLINE :

`GLINE` is used to draw a line in the "image" G$, starting from *x* and length *size*. `GLINE` can be used to plot nice patterns. For example :

```
first  size     gap
<--><--------><------>              Length
<---------------------------------------------------->
_______________________________________________________
|     xxxxxxxxxx       xxxxxxxxxx      xxxxxxxxx
|   xxxxxxxxxx       xxxxxxxxxx      xxxxxxxxx
| xxxxxxxxxx       xxxxxxxxxx      xxxxxxxxx
|
```

Each line is build by specifying 5 parameters :

1) the coordinate of the fist point : *x* (AFTER LEFT MARGIN)
2) the line length : *length* (IN DOTS; MAX = 6")
3) the first gap : *first*
4) the size of a dash : *size*
5) the gap between dashes : *gap*.

To draw a pattern similar to this one, you only have to set *first* as follows :

first = MOD (*first* − 2 , *gap*)

To draw a straight line, use :

first = *size* = 1
gap = 0

References

JPC 35 (page 38) `GLINE` and GRAPH subprogram by Pierre David.

JPC 42 (page 29) sample of use by Eric Gengoux.

HP-71 Graphic Module User's Manual by Pierre David.

HMS

HMS (Hour Minute Second) converts decimal hour or degree data into an equivalent value in HMS format.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
HMS  ( argument  )
```

Example

`A=HMS(121.5)`

An angle of 121.5 decimal degrees is equivalent to an angle of 121 degrees 30'.

Input Parameter

Item	Description	Restrictions
argument	Numeric expression.	None.

Operation

The HMS format :

The functions HMS, HR, HMSADD and HMSSUB manipulates time or angle data in the HMS format. The arguments must be real numbers.

The HMS format is *h.mmssd*, where :

- *h* = 0 or more digits representing the integer part of the number,
- *mm* = 2 digits representing the number of minutes,
- *ss* = 2 digits representing the number of seconds, and
- *d* = 0 or more digits representing the fractional decimal part of seconds.

These numbers are not related to the current angle mode setting (see `OPTION ANGLE`).

The HMS function :

HMS converts decimal hour or degree data into an equivalent value in HMS format.

References

JPC 25 (page 52) first version by Michel Martinet.

HMS (continued)

JPC 50 (page 29) second version by Guy Toublanc.

HP-41 Owner's Manual.

HP-28C Reference Manual.

Related Keywords

HMSADD, HMSSUB, HR, trigonometric functions

Authors

Michel Martinet and Guy Toublanc.

HMSADD

HMSADD (Hour Minute Second ADD) returns the sum of two arguments interpreted using the HMS format.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
HMSADD ( arg1 , arg2 )
```

Example

`A=HMSADD(12.3456,-20.1721)`	Store in A the sum of 12 hours 34'56" and -20 hours 17'21". The sum is -7 hours 42'25".

Input Parameters

Item	Description	Restrictions
arg1, arg2	Numeric expressions interpreted according to the HMS format.	None.

Operation

HMSADD adds two arguments interpreted as sexagesimal numbers in HMS format. The value returned is also in the HMS format.

See the HMS function for more informations about the HMS format.

References

JPC 25 (page 52) first version by Michel Martinet.

JPC 50 (page 29) second version by Guy Toublanc.

HP-41 Owner's Manual.

HP-28C Reference Manual.

HMSADD was called HMS+.

Related Keywords

HMS, HMSSUB, HR, trigonometric functions

HMSADD (continued)

Authors

Michel Martinet and Guy Toublanc.

HMSSUB

HMSSUB (Hour Minute Second SUBtract) returns the difference of two arguments interpreted using the HMS format.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
HMSSUB ( arg1 , arg2 )
```

Example

`A=HMSSUB(12.3456,20.1721)`	Stores into variable A the difference between 12 hours 34'56" and 20 hours 17'21".

Input Parameters

Item	Description	Restrictions
arg1, arg2	Numeric expressions interpreted using the HMS format.	None.

Operation

HMSSUB subtracts two arguments interpreted as sexagesimal numbers in HMS format. The value returned is also in the HMS format.

See function HMS for more informations about HMS format.

References

JPC 25 (page 52) first version by Michel Martinet.

JPC 50 (page 29) second version by Guy Toublanc.

HP-41 Owner's Manual.

HP-28C Reference Manual.

HMSSUB was called HMS-.

Related Keywords

HMS, HMSADD, HR, trigonometric functions

Authors

Michel Martinet and Guy Toublanc.

HR (HouR) converts a number from HMS format to its decimal equivalent.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
HR ( argument )
```

Example

```
A=HR(121.3)
```

121 degrees 30' correspond to 121.5 decimal degrees.

Input Parameter

Item	Description	Restrictions
argument	Numeric expressions interpreted using the HMS format.	None.

Operation

HR converts real numbers in the HMS format to real numbers in the decimal format which can be used directly by trigonometric functions. HR is the inverse of HMS.

See the HMS function for more informations about the HMS format.

References

JPC 25 (page 52) first version by Michel Martinet.

JPC 50 (page 29) second version by Guy Toublanc.

HP-41 Owner's Manual.

HP-28C Reference Manual.

Related Keywords

HMS, DEGREES, HMSADD, HMSSUB, trigonometric functions

Authors

Michel Martinet and Guy Toublanc.

HTA$ (Hexadecimal To Ascii) converts a string of hexadecimal digits into an Ascii character string.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
HTA$  (  hexadecimal string  )
HTA$  (  hexadecimal string  ,  mode  )
```

Examples

`A$=HTA$("0516279637")`	Stores the string "Paris" into variable A$.
`A$=HTA$("5061726973",1)`	Stores the string "Paris" into variable A$.

Input Parameters

Item	Description	Restrictions
hexadecimal string	String expression.	Even number of hexadecimal digits either uppercase or lowercase.
mode	Numeric expression. Default : 0	None.

Operation

HTA$ is the inverse function of ATH$.

As for ATH$, interpretation of hexadecimal digits depends on the *mode* parameter :

If *mode* = 0, logical value *false* (default), the hexadecimal string is built by reversing the order of digits in a character. For example, the string "14" represents character "A". This representation is similar to the internal data format in the HP-71.

If *mode* is not 0, logical value *true*, a standard representation is used : the string "41" represents character "A".

References

JPC 22 (page 31) first version by Michel Martinet.

JPC 27 (page 34) second version by Michel Martinet.

To be published : third version by Pierre David.

Related Keywords

ATH$, ASC$

Authors

Pierre David and Michel Martinet.

IF ... THEN ... ELSE ... END IF

The structure `IF` ... THEN ... ELSE ... END `IF` extends the standard structure to allow multiple line statements.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
IF logical expression THEN
   program segment
END IF
or :
IF logical expression THEN
   program segment
ELSE
   program segment
END IF
```

Examples

```
100 IF X=0 THEN
110   INPUT X
120 ELSE
130   X=X+1
140 END IF
150  :
```

If X is 0 then INPUT is executed, otherwise X is incremented.

```
100 IF X=0 THEN
110   INPUT X
120 END IF
130  :
```

If X is 0 then INPUT is executed, otherwise execution continues after line 130.

Input Parameters

Item	Description	Restrictions
logical expression	Numeric expression evaluated as true if non-zero and false if zero..	None.
program segment	Any number of contiguous lines.	None.

Operation

The structure `IF` ... ELSE ... END `IF` (or `IF` ... END `IF`) of JPC Rom extends the standard `IF` structure.

The difference occurs when there is a statement after the THEN clause i.e. an end of line, a character (`@`, a comment (`!`), or an ELSE that is not preceded by an `IF` ... THEN on the same line.

IF ... THEN ... ELSE ... END IF (continued)

For example :

```
10 IF X=0 THEN                    JPC Rom IF
10 IF X=0 THEN @ BEEP                "
10 IF X=0 THEN ! Remarque            "

10 IF X=0 THEN 'BEEP'             standard IF
10 IF X=0 THEN 10                    "
10 IF X=0 THEN BEEP                  "
```

Programs with standard IF can be executed without JPC Rom in your HP-71.

If the evaluation of *logical expression* is true (different from zero), execution resumes at the first statement following THEN. If ELSE is present, execution will continue after END IF skipping the block of ELSE code.

If the evaluation of *logical expression* is false (null) execution resumes immediately after the ELSE clause, if any, or after the END IF instruction.

Program segments can contain any loop structure. These structures must be properly matched otherwise the error `JPC ERR:Structure Mismatch` is reported.

References

JPC 52 first version by Pierre David and Janick Taillandier.

HP 9000 series 200/300 Basic 4.0.

Related Keywords

IF ... THEN, SELECT ... END SELECT

Authors

Pierre David and Janick Taillandier.

INVERSE

`INVERSE` displays the binary complement of the contents of the display.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
INVERSE
INVERSE begin , end
```

Examples

```
IF KEYDOWN("I") THEN INVERSE
```
Inverts the display if key [I] is pressed.

```
INVERSE 10,121
Inverts the display from column 10 to column 121, i.e. leaving
10 column unchanged on both sides.
```

Input Parameters

Item	Description	Restrictions
begin	Numeric expression rouded to an integer. Default : 0	0 through 131
end	Numeric expression rouded to an integer. Default : 131	begin through 131

Operation

`INVERSE` inverts the whole LCD display or a part of it : black dots turns to white and white ones to black. The contents of the display buffer remain unchanged.

The contents of the screen memory (returned by `DISP$`) is not modified.

The optional parameters specify the first and last column for the inversion.

References

JPC 19 (page 25) Forth word to invert the display by Jean-Pierre Bondu.

JPC 24 (page 37) first version by Jean-Jacques Moreau.

JPC 25 (page 59) Forth word to return the address of a graphic column by Jean-Pierre Bondu.

INVERSE (continued)

To be published : second version by Pierre David and Janick Taillandier.

Internal Design Specification Volume I, chapter 3.2.1.

Related Keywords

`GDISP$`, `GDISP`, `INV$`

Authors

Pierre David, Jean-Jacques Moreau and Janick Taillandier.

KA is an interactive address directory editor.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
KA
KA file
```

Examples

`KA`	Enters the address directory editor using the default file ADRS.
`KA EXAMPLE`	Enters the address directory editor using the EXAMPLE address file.

Input Parameter

Item	Description	Restrictions
file	String expression or unquoted string. Default : ADRS	File name with optional device specifier.

Operation

Getting started

Let us begin with a little guided tour of KA functions. We will first enter 3 addresses (or card), then we will learn how to browse through the file, search it and modify cards.

First, type KA to enter the address directory management function. KA tells you that the file is empty (`Empty file` message) and is waiting for your commands.

To enter the first address, press the [f] key (yellow prefix key) and [INPUT] (corresponding to the G key).

Input the name and first name separated by a slash (/). For example :
`Selere/Jacques` then [ENDLINE]

Now, you can enter the phone number :
`(3) 14 15 92 65` then [ENDLINE]

Now the address itself. We have 4 lines at our disposal, but we will use only 2 lines in this example :

```
33, rue des Marguerites        then [ENDLINE]
75028 Paris                    then [ENDLINE]
```

We want to exit address input. We still have 2 empty address lines, a general purpose note line and a criterion line. Do not input anything in these lines. Press [ENDLINE] 4 times, and one more time to validate the input.

As an address directory with a single address is rather poor, let us enter 2 new addresses. It is up to you (don't forget to press [f] [INPUT] first) :

Caux/Harry
(2) 71 18 28 18
14 bd des Paquerettes
27085 Paris

and

Breille/Jean
(0) 69 31 47 18
1, allée du Mimosa
82705 Paris

Now, you have three adresses in your file (press [f] [CAT] continuously to check this) we will see how to move around the file.

Try cursor keys [↑] et [↓] to scroll a card. You will see empty lines corresponding to those you have entered previously. You can go to the first or last line by using cursor keys prefixed by the [g] key (blue prefix key).

If a line is longer than 22 characters you can read it by scrolling the display with [→] and [←] keys (or these keys prefixed by the [g] key).

To skip from card to card, you have to use the [(] and [)] keys (parenthesis), or these keys prefixed by the blue key [g] to go from one file end to the other. Note that cards are sorted in alphabetical order.

Go to the beginning of the file (key [g] [(]). We will search Jacques Selere's address. Press the [S] key, then [E] [L] [E] [R] [E] and press [ENDLINE].

Now you are on the card for Jacques Selere. Not that the search does not make differences between uppercase and lowercase characters.

If you want to find the card whose name begins with " BR ", press the [B] key and then type characters R and . (dot) and press [ENDLINE]. This is the generic search. REV X: VICE VERSA! USE A DOT ONLY FOR EXACT SEARCHES.

If you want to delete the card corresponding to Jacques Selere. Go to this card and press [f] [DELETE]. A confirmation is requested, if you press [Y] the card is deleted.

Finally we will modify a card. Go to Harry Caux's card and press [f] [EDIT]. Now, you can modify each field in the card. You exit this mode and store the modifications by pressing the [RUN] key or pressing twice the [ENDLINE] key on the last line.

Address files

KA is an interactive address directory manager. Addresses are stored in a file whose type is ADRS. The default address file is called ADRS ; it is automatically searched when KA is executed without parameter.

When the specified file is on an external mass storage device, such as a disk drive, KA first copy the file into Ram and then process the data contained in it.

Addresses

An address file contains cards, each one contains an address other optional informations. A card is identified by the name associated to it.

There is no other limit to the number of cards that KA can manage than the memory available in your HP-71.

Each card is made up of 8 lines, organized in the following way :
- name and first name, separated by a /,
- phone number,
- 4 lines to store the address,
- a line to store general informations, and
- a line to sore a criterion to be used by your own programs.

The first line contains the name and first name, separated by a slash (/). KA will add it for you if you forget it.

Each line can be up to 90 characters long.

Using KA

When you enter KA, you are in browse mode. In this mode you can look at all your addresses.

In case of difficulty, press [ATTN]. This exits KA in *browse* mode or return to *browse* mode from other ones.

Browse mode

In this mode you can skip from card to card using [<] and [>] keys. The [g] [<] and [g] [>] keys are used to go to the beginning and the end of the file respectively.

You can scroll through a card using [↑] and [↓] keys. You can also use the [g] [↑] and [g] [↓] keys to go to the beginning and the end of the card respectively.

Keys [0] through [7] allows you to go directly to a specified line inside the card.

The [f] [CAT] (kept pressed) displays the number of cards in the file and an estimate of the number of cards than can be input. Please note that this figure is an estimation based on the average size of the cards and on the memory available in main \The [f] [DELETE] is used to remove a card. A confirmation is asked. The card will be erased only if you answer [Y] (*yes*).

Editing mode

The *editing* mode allows you to modify a card ([f] [EDIT] key) or to enter a new one ([f] [INPUT] key).

From *browse* mode, you enter *editing* mode by pressing :
- [f] [EDIT] (you edit the current card), or
- [f] [INPUT] (the edited card is empty).

Then, you can modify or input the edited card. The following keys are valid :

- [↑], [↓], [g] [↑] or [g] [↓] : go to another line in the card without validation of the current line,
- [ENDLINE] : validation of the current line and skip to next line,
- [ATTN] : clears current line,
- [ATTN] twice : exits the *editing* mode without entering the modifications.
- [ENDLINE] twice on the last line or [RUN] : validates the card and stores it in the file.

When validated, the card is automatically inserted in the file in alphabetical order.

Search mode

The *search* mode allows you to search for a name in the whole file. From *browse* mode, simply press one of the keys from [A] through [Z] to input the first character of the name. Then you can type the remaining letters.. When you press [ENDLINE], the name (and first name if typed) are searched for in the file. The error `Not Found`) is reported if the name was not found.

In *search* mode you have 3 kind of search :

- name search :
When you enter the name only (i.e. no / character), the name is searched in the file from the current file to the end of file and from the beginning to the current card. The first name found becomes the current card.

- name and first name search :
the search path is the same as before. if a card has the same name but a different first name it will not be found.

- generic search :
This is the most useful search kind. You have only to type the beginning of the name and a . (dot). KA will search the first card whose name begins with the requested characters. If no card match, the next one is edited.

No difference is made between uppercase and lowercase characters during the search.

The generic search is easier and faster to use. This is the preferred search method in day to day use of KA.

Password

Each address file can have a password. If it has one, each time you enter KA you will be prompted for the password and you will have to provide it if you use the programmable keywords to access the file (see ADCREATE).

You can change the file password (it is the only way to do it) by pressing the [f] [KEY] key in *view* mode. You are asked for a new password.

- if you pressl[ATTN], the password is left unchanged,
- if you do not enter anything and press [ENDLINE], the password is cleared, and finally
- if you type a new password (up to 8 characters) and press [ENDLINE], it is stored.

If you use an HP-Il video interface, the password will not be displayed on the screen when you type it.

References

AGENDA program for the HP-75 user's manual.

Related Keywords

ADCREATE, ADDELETE, ADFIND, ADGET, ADPUT, ADSIZE

Author

Pierre David

NOTE: PASSWORD IS A CONVENIENCE FEATURE ONLY.
IT IS NOT ENCRYPTED IN THE ADRS FILE IN ANY WAY.

KEYWAIT$

KEYWAIT$ waits until a key is pressed and then returns a string representing its keycode.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
KEYWAIT$
```

Examples

```
A$=KEYWAIT$
```

Waits for a key pressed and returns the keycode into A$.

```
10 LOOP
20   DISP KEYWAIT$
30 END LOOP
```

Loops and displays all key pressed.

```
10 SELECT KEYWAIT$
20   CASE "A" TO "Z"
30     DISP "LETTER"
40   CASE "#46"
50     DISP "[RUN]"
60   CASE ELSE
70     DISP "OTHER"
80 END SELECT
```

Waits for a key pressed, then displays "LETTER" if the key was a letter, "[RUN]" if it was the [RUN] key, "OTHER" otherwise.

Operation

KEYWAIT$ places the HP-71 into a low-power state until a key is pressed.

The keycode returned in the string is in the same format as the standard function KEY$.

Using KEYWAIT$ allows you no to write loops to wait for a character like this one :

```
10 K$=KEY$ @ IF K$="" THEN 10
```

References

JPC 20 (page 50) KEYWAIT$ listing by Pierre David and Michel Martinet.

Related Keywords

ATTN, FKEY, KEY$, PUT

Author

Hewlett-Packard

LEAVE

LEAVE exits from a structured programming loop, see WHILE, REPEAT or LOOP.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
LEAVE
```

Example

`IF I=INF THEN LEAVE`	Exits the loop if I is infinite.

Operation

LEAVE allows early exit from WHILE ... END WHILE or REPEAT ... UNTIL loops.

LEAVE is the only way to exit from a LOOP ... END LOOP structure.

References

JPC 31 (page 38) first version by Janick Taillandier.

JPC 52 second version by Pierre David and Janick Taillandier.

Related Keywords

EXIT, LOOP ... END LOOP, REPEAT ... UNTIL, WHILE ... END WHILE

Authors

Pierre David and Janick Taillandier.

RENAMED LXON and LXOFF (REV X)

LEX enables or disables a Lex file.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

~~LEX *file* OF~~F LXOFF Filename
~~LEX *file* ON~~ LXON filename

Examples

`LEX STRINGLX OFF`	Disables the LEX file STRINGLX. Functions in this Lex are no more available. The file type is changed to D-LEX.
`LEX STRINGLX ON`	Enables the D-LEX file STRINGLX. Functions in this Lex are again available.

Input Parameter

Item	Description	Restrictions
file	String expression or unquoted string.	File name with optional device specifier.

Operation

Disabling Lex files

The processing speed of the HP-71 is directly related to the number of Lex files in memory. The more files, the slower the machine. It is better to have one big Lex file with 90 functions than 9 files with 10 functions each. If your HP-71 is loaded with many small Lex files, you may wish to disable some, and enable them back when you need them.

Also, if Lex files contain two keywords with the same identification (ID) and token, you can use either keywords disabling the Lex you do not wish to use.

Files in Rom or Eprom cannot be enabled.

The LEX ON/OFF command

LEX *file* OFF change the type of *file* from LEX to D-LEX, so that it is no more cative. Its functions cannot be used any more until a LEX *file* ON is executed on this file.

D-LEX files

Disabled files are listed with a D-LEX type during a CAT or CAT$ as long as JPC Rom is plugged in your HP-71. These files can be copied to a mass storage but can be copied back to Ram only if JPC Rom is plugged in the HP-71.

References

JPC 24 (page 30) first version by Michel Martinet.

To be published : new version by Pierre David and Janick Taillandier.

LEX was called ENABLE and DISABLE

Related Keywords

EDIT, MERGE

Authors

Pierre David, Michel Martinet and Janick Taillandier.

NOTE: THE HELPF COMMAND IN HLPLEX
CAN BE USED ON DISABLED LEXFILES!

LOOP ... END LOOP

LOOP ... END LOOP defines an endless loop.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
LOOP
   program segment
END LOOP
```

Example

```
10 INPUT X
20 LOOP
30   DISP X;X*X
40   X=X+1
50 END LOOP
```

Defines an endless loop displaying a sequence of numbers and their squares. The only way to stop this loop is by pressing [ATTN].

Input Parameter

Item	Description	Restrictions
program segment	Any number of contiguous lines.	None.

Operation

The structure LOOP ... END LOOP allows endless looping with conditional exits through LEAVE.

The repeated code segment begins after the keyword LOOP and ends before the keyword END LOOP. When you reach END LOOP execution branches to the first statement following LOOP.

The program segment can contain any number of LEAVE. The only restriction is that LEAVE instructions may not be placed inside nested structures. (IT'S OKAY, BUT IT ONLY LEAVES THE INNER LOOP)

Program segments may contain any kind of loop structure. However, these structures must be properly matched, or the error `JPC ERR:Structure Mismatch` will be reported.

References

JPC 52 first version by Pierre David and Janick Taillandier.

HP 9000 series 200/300 Basic 4.0.

LOOP ... END LOOP (continued)

Related Keywords

ATTN, LEAVE, FOR ... NEXT, WHILE ... END WHILE, REPEAT ... UNTIL

Authors

Pierre David and Janick Taillandier.

MAP

MAP applies a mapping function to the contents of a text file.

■ Statement	■ Keyboard Execution
O Function	O CALC Mode
O Operator	■ IF...THEN...ELSE
	■ Device Operation

```
MAP file , string1 , string2
MAP file , string1 , string2 , from
MAP file , string1 , string2 , from , to
MAP # channel , string1 , string2
MAP # channel , string1 , string2 , from
MAP # channel , string1 , string2 , from , to
```

Example

```
MAP TOTO,"EQ","eq",10,50
```

Replaces, in file TOTO, uppercase characters "E" and "Q" by the lowercase characters "e" and "q" respectively, from record 10 to record 50.

Input Parameters

Item	Description	Restrictions
file	String expression or unquoted string.	File name with optional device specifier. File must reside in Ram or independent Ram.
channel	Numeric expression rounded to an integer.	1 through 255.
string1, string2	String expression.	Both strings must have the same length.
from	Numeric expression rounded to an integer (first record). Default : 0	0 through 1048575.
to	Numeric expression rounded to an integer. Default : Last record.	0 through 1048575.

Operation

MAP scans the file specified by *file* or by its associated channel number (*channel*).

Each character is tested to check if it is included in *string1*. If so, the corresponding character in *string2* replaces the original one.

If a character occurs in *string2* twice, the second occurrence will never be used, as the matching process always starts at the beginning of *string2*.

References

STaK (Finnish Users Club Journal) November 1986. First version by Tapani Tarvainen.

JPC 46 (page 18) translation of Tapani Tarvainen's article in *JPC*

Related Keywords

`ASSIGN #`, `MAP`, `ROMAN`

Author

Tapani Tarvainen

MAP$

MAP$ applies a mapping function to the contents of a character string.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
MAP$ ( string1 , string2 , string3 )
```

Example

```
MAP$ ("strings","s","S")
```
Returns "StringS".

Input Parameters

Item	Description	Restrictions
string1	String expression.	None.
string2, string3	String expressions.	Both strings must have the same length.

Operation

MAP$ scans all characters in *string1* ; each character is matched against the characters in *string2* and, if a match is found, the character is replaced by the corresponding character in *string3*.

If a character occurs in *string2* twice, the second occurrence will never be used, as the matching process always starts at the beginning of *string2*.

References

STaK (Finnish Users Club Journal) November 1986. First version by Tapani Tarvainen.

JPC 46 (page 18) translation of Tapani Tarvainen's article in *JPC*

Related Keywords

MAP, REPLACE$, ROMAN

Author

Tapani Tarvainen

MARGIN

MARGIN enables a beep when the cursor reaches the specified position.

■ Statement	■ Keyboard Execution
O Function	O CALC Mode
O Operator	■ IF...THEN...ELSE
	■ Device Operation

```
MARGIN
MARGIN position
```

Examples

`MARGIN 80`	A beep occurs when the cursor reaches column 80.
`MARGIN`	Disables the previous MARGIN setting.

Input Parameter

Item	Description	Restrictions
position	Numeric expression rounded to an integer. Default : 0	0 through 96.

Operation

MARGIN stores the specified cursor position. Thereafter, your HP-71 will beep every time the cursor reaches this position assuming the beep mode has been enabled (BEEP ON), and that JPC Rom is in your HP-71.

MARGIN is active during normal input as well as during INPUT, LINPUT or FINPUT in program mode, or in FORTH with Forth / Assembler module or Translator Pac.

MARGIN without parameter or MARGIN 0 disable this feature and recovers the Ram used to store the cursor position (4.5 bytes).

References

JPC 26 (page 33) first version by Michel Martinet.

To be published : modifications to use HP allocated resources, by Pierre David and Janick Taillandier.

HP-75 MARGIN statement.

MARGIN (continued)

Related Keywords

`FINPUT`, `FORTH`, `INPUT`, `LINPUT`

Authors

Pierre David, Michel Martinet and Janick Taillandier.

MAXD

MAXD (MAXimum Directory size) returns the maximum number of entries that can be stored in the directory of a mass storage medium.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	○ Device Operation

```
MAXD ( device specifier )
```

Examples

`A=MAXD(".DISK")`	Stores into variable A the number of entries in the directory of medium with label ".DISK".
`DISP MAXD(A$)`	Displays the number of entries available on the medium specified by the contents of A$.
`N=MAXD(D)`	Stores into N the number of entries on the disk at address D in the loop.

Input Parameter

Item	Description	Restrictions
device specifier	See standard HP-IL definitions.	Unquoted strings are not allowed.

Operation

Mass storage :

Peripherals recognized as mass storage by the HP-71 use the *Filbert* protocol. They include the cassette drive HP-82161 and the disk drive HP-9114. They can be specified using `:TAPE` or `:MASSMEM`.

The standard unit for mass storage operations is the *sector*. Each *sector* has a storage capacity of 256 bytes ; this is the basic element for transfers between the mass storage device and the controller (the HP-71).

A digital cassette for the HP-82161 has 512 sectors, or 131 Kbytes. A double-sided disk for the HP-9114 has 2464 sectors or 630 Kbytes.

Information is stored on mass storage media as follows :

- *system data* : the HP-IL controller (the HP-71) uses it to store informations such as initialization date and time, volume label, directory size and medium size. This use sectors 0 and 0, or 512 bytes.

- *directory* : this is a table of information about the files, such as creation date and time, location on the medium, size, type, etc. Each directory entry consists of 32 bytes. The directory normally starts at sector 2.

- *file storage area* : this is where the files are stored.

The directory space is allocated during medium initialization, using the `INITIALIZE` statement. The information is stored in the system data sectors. The real directory size is always the smallest multiple of 8 greater than the specified number in order to fill an integer number of sectors.

By default, `INITIALIZE` uses a directory size equal to 1/32th the medium size (in sectors). So, the number of entries is :
entries = medium size in bytes / 1024

The MAXD function :

MAXD returns the maximum number of directory entries on the specified medium.

This number is always a multiple of 8. To get the space used by the directory, you can do :

sectors : `MAXD / 8`
bytes : `MAXD * 32`

Interrupting the function with [ATTN] :

This function can be interrupted by pressing the [ATTN] key twice. This causes the message `HPIL ERR:Aborted`. It may then be necessary to execute `RESTORE IO` to reactivate the HP-IL system.

References

JPC 30 (page 40) first version by Michel Martinet.

HP82161 Digital Cassette Drive Owner's Manual.

HP-IL Interface Owner's Manual Chapter 3 and appendix D.

Related Keywords

`INITIALIZE`, MEMD, MAXM, RREC$, WREC

Author

Michel Martinet

MAXM

MAXM (MAXimum Medium capacity) returns the maximum storage capacity available on the medium.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	○ Device Operation

```
MAXM ( device specifier )
```

Example

`A=MAXM("%16")`	Returns the space, in bytes, of the medium.

Input Parameter

Item	Description	Restrictions
device specifier	See standard HP-IL definitions.	Unquoted strings are not allowed.

Operation

MAXM returns the capacity of the mass storage device specified by *device specifier*. This capacity includes the system sectors and the sectors allocated to the directory.

To get the space available for user data, you can do :

```
MAXM(D)-512-MAXD(D)*32
```

For more information, see MAXD and the HP-IL interface owner's manual.

Interrupting the function with [ATTN] :

This function can be interrupted by pressing the [ATTN] key twice. This causes the message `HPIL ERR:Aborted`. It may then be necessary to execute `RESTORE IO` to reactivate the HP-IL system.

References

JPC 30 (page 40) first version by Michel Martinet.

HP82161 Digital Cassette Drive Owner's Manual.

HP-IL Interface Owner's Manual Chapter 3 and appendix D.

MAXM (continued)

Related Keywords

MAXD, MEMM, RREC$, WREC

Author

Michel Martinet

MDY

MDY (Month Day Year) enables date input in numeric format *mm.ddyyyy*.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
MDY
```

Example

```
IF K$="N" THEN MDY
```

If K$ equals "N", then use U.S. date format.

Operation

In the mode enabled by DMY, date parameters used by JPC Rom date functions can be input using a numeric *mm.ddyyyy* format.

The MDY format is the default format after a memory reset.

Date information can always be entered using the string format, which is not affected by the MDY / DMY modes.

For more information about date input formats, see the DATESTR$ function.

References

JPC 28 (page 40) first version by Laurent Istria.

JPC 49 (page 24) second version by Pierre David and Janick Taillandier.

Related Keywords

DATESTR$, DATEADD, DDAYS, DMY, DOW, DOW$

Authors

Pierre David, Laurent Istria and Janick Taillandier.

MEMD

MEMD (MEMory in Directory) returns the number of entries in the directory of the specified medium that remain available for new files.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	○ Device Operation

```
MEMD  ( device specifier )
```

Example

`A=MEMD(":TAPE")`	Returns the number of entries available in the directory..

Input Parameter

Item	Description	Restrictions
device specifier	See standard HP-IL definitions.	Unquoted strings are not allowed.

Operation

Purged files :

When files are removed from the medium using the PURGE command, the corresponding entry in the directory becomes available for new files. This creates gaps in the directory that are invisible to the user.

These gaps may occasionally become too numerous. When this happens, a PACKDIR may be necessary to pack the directory area, and remove gaps.

The MEMD function :

MEMD returns the number of entries available in the directory. This count includes purged file entries, if any.

MEMD considers purged files entries as available : MEMD acts as if a PACKDIR had occurred.

Interrupting the function with [ATTN] :

This function can be interrupted by pressing the [ATTN] key twice. This causes the message HPIL ERR:Aborted. It may then be necessary to execute RESTORE IO to reactivate the HP-IL system.

References

JPC 30 (page 40) : first version by Michel Martinet.

HP82161 Digital Cassette Drive Owner's Manual.

HP-IL Interface Owner's Manual Chapter 3 and appendix D.

Related Keywords

`MAXD`, `MEMM`, `PACKDIR`, `RREC$`, `WREC`

Author

Michel Martinet

MEMM

MEMM (MEMory on Medium) returns the available room in the file storage area of the specified medium.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	○ Device Operation

```
MEMM ( device specifier )
```

Example

`A=MEMM(":HP9114")`	Returns the storage capacity remaining available in the first HP-9114 unit on the loop.

Input Parameter

Item	Description	Restrictions
device specifier	See standard HP-IL definitions.	Unquoted strings are not allowed.

Operation

Purged files :

When files are purged from a medium using the PURGE command, the corresponding space in the file storage area becomes available. This produces gaps on the medium invisible to the user.

However, these gaps may become too numerous. In this case a PACK may be necessary to pack the directory and file storage areas, and remove these gaps.

The MEMM function :

The MEMM function returns the storage capacity in the file storage area that is available for new files.

MEMM includes the space reserved by purged file entries : MEMM gives the same result as if a PACK had occurred.

Interrupting the function with [ATTN] :

This function can be interrupted by pressing the [ATTN] key twice. This causes the message `HPIL ERR:Aborted`. It may then be necessary to execute RESTORE IO to reactivate the HP-IL system.

MEMM (continued)

References

JPC 30 (page 40) : first version by Michel Martinet.

HP82161 Digital Cassette Drive Owner's Manual.

HP-IL Interface Owner's Manual Chapter 3 and appendix D.

Related Keywords

MAXM, MEMD, PACK, RREC$, WREC

Author

Michel Martinet

MENU

MENU is an interactive menu facility.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

MENU (*number of elements*)
MENU (*number of elements* , *first element*)

Example

```
10 ATTN OFF @ M=1
20 DATA ONE,TWO,THREE,FOUR
30 RESTORE 20
40 M=MENU(4,M) @ ON M GOTO 60,70,80,90
50 ATTN ON @ BEEP @ END
60 DISP "ONE" @ GOTO 30
70 DISP "TWO" @ GOTO 30
80 DISP "THREE" @ GOTO 30
90 DISP "FOUR" @ GOTO 30
```

Display a choice of 4 items, scrolled using vertical cursor keys. The user validates his choice with [ENDLINE], or exits with [ATTN] and resumes execution at line 40.

Input Parameters

Item	Description	Restrictions
number of elements	Numeric expression rounded to an integer.	0 through 1048575.
first element	Numeric expression rounded to an integer. Default : 1	0 through 1048575.

Operation

MENU provides interactive menu processing for programs.

MENU uses items stored in DATA statements, starting with the data item at the current DATA pointer position.

The various menu items can be scrolled using vertical cursor keys [^], [v], [g] [^] et [g] [v].

A choice is validated using [ENDLINE]. The sequence number of the corresponding element is returned. The first item on the menu list returns a value of 1 and the last item returns *number of elements*.

Pressing [ATTN] exits MENU and skips to next program line (not next statement). This allows special processing of user interrupts.

MENU (continued)

If *first element* is specified, it represents the number of the first data element displayed. Keys [^] and [g] [^] allows to scroll to previous elements.

References

JPC 26 (page 34) first version by Jean-Jacques Dhénin.

Related Keywords

DATA, FINPUT, READ, RESTORE

Author

Jean-Jacques Dhénin

MERGE

MERGE extends the standard MERGE function to Lex files. MERGE is nonprogrammable.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
MERGE file
MERGE file , first line
MERGE file , first line , last line
```

Example

`EDIT STRINGLX @ MERGE KEYWAIT`	Chain Lex file KEYWAIT into STRINGLX.

Input Parameters

Item	Description	Restrictions
file	String expression or unquoted string.	File name with optional device specifier.
start line	Integer constant. Default : First program line.	1 through 9999 (no meaning for Lex files).
final line	Integer constant. Default : Start line.	Start line through 9999 (no meaning for Lex files).

Operation

Merging Lex files :

Merging Lex files links two or more Lex files into a single file in Ram or independent Ram.

This process has many advantages. First, it allows you to bring together all the keywords and operating system enhancements required by any given application package in a single file. This can greatly simplify the application main program, since it needs to verify the presence of only one Lex file instead of several. Also, the time required to load the combined Lex file is significantly less than the time required to load all the component Lex files.

Linking existing Lex files is much easier than writing a new special purpose Lex with all the prerequisite capabilities. It helps conserve HP-71 system resources, since you can link files with different Id or with non-consecutive token numbers. Finally, it reduces memory requirements by 18.5 bytes for each file that is merged, since linked Lex files share a common 18.5 bytes header.

MERGE (continued)

Keep in mind that linking Lex files does not reduce the number of entries in the configuration buffers, and that the number of poll handlers also remains the same. This means that linking Lex files will not result in the improved system performance that you get from combining the source code of the component Lex files to produce a single Lex such as JPC Rom.

You should link (merge) Lex files for the same reasons you merge several subprograms into a single file : convenience and simplified mass storage management.

Using MERGE :

To merge Lex files F1, F2 and F3 into file F1, you have to execute :

```
EDIT F1
MERGE F2
MERGE F3
EDIT
```

`EDIT F1` makes F1 the current *workfile*. MERGE `F2` merges F1 into F2. MERGE `F3` merges F3 into F1 (which is now F1 + F2).

The final `EDIT` restores `workfile` as the current file.

F2 and F3, which are still in memory, may now be purged as after a standard MERGE of Basic or Keys files.

If you want to restores the current file, you can use :

```
A$=CAT$(0)[1,8]
EDIT F1
MERGE F2
MERGE F3
EDIT
```

Warning !

Never purge the current Lex file !

A bug seems to exist. The Basic statement PURGE does not work properly if the current file is a Lex : the file pointer is not reset to the `workfile`. This typically causes strange CAT operation. In most cases, a simple `EDIT` corrects the problem.

References

JPC 23 (page 47) Basic program to merge Lex files by Michel Martinet.

JPC 37 (page 22) assembly language Lex file merger by Pierre David and Michel Martinet.

Related Keywords

`EDIT`, MERGE

Authors

Pierre David and Michel Martinet.

MODE

MODE sends an escape sequence that changes the print pitch on the PRINTER IS device.

■ Statement	■ Keyboard Execution
O Function	O CALC Mode
O Operator	■ IF...THEN...ELSE
	O Device Operation

```
MODE argument
```

Example

`MODE 2`	Sets the printer to the compressed print mode.

Input Parameter

Item	Description	Restrictions
argument	Numeric expression rounded to an integer.	0 to 999.

Operation

MODE changes the print pitch on the peripheral selected by the last PRINTER IS command.

On a ThinkJet printer,

MODE 0 selects 80 characters per line,
MODE 1 selects 40 characters per line,
MODE 2 selects 142 characters per line and
MODE 3 selects 71 characters per line.

Codes sent to the printer :

MODE *n* : `ESC & k` *n* `S`

References

JPC 26 (page 39) first version by Pierre David.

JPC 40 (page 16) second version by Pierre David.

Also consult your printer reference manual.

Related Keywords

BOLD, PRINTER IS, UNDERLINE, WRAP

Author

Pierre David

NEXTOP$

NEXTOP$ (NEXT OPcode) returns the address of the next assembler instruction.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
NEXTOP$ ( hexadecimal address pointer )
```

Examples

```
A$=NEXTOP$("0BD38")
```

Stores 0BD3C, the address of the first instruction of the POP1S routines (at address 0BD38), into A$.

```
10 A$="00000"
20 LOOP
30    DISP OPCODE$(A$)
40    A$=NEXTOP$(A$)
50 END LOOP
```

This little program provides an automatic disassembler, starting from address 00000. Each instruction mnemonic is displayed, then the address is updated.

Input Parameter

Item	Description	Restrictions
hexadecimal address pointer	String expression containing hexadecimal digits.	Up to 5 uppercase or lowercase digits.

Operation

NEXTOP$ returns the address of the opcode that follows the one located at the hexadecimal address pointer you specify. Together with OPCODE$, this function allows you to easily disassemble HP-71 machine language.

The current instruction length (in nibble) can be computed by subtracting the current address from the address returned by NEXTOP$:
len = HTD(NEXTOP$(*current address*))-HTD(*current address*)

Warning : if the hexadecimal address points to a data field rather than to a machine language instruction, the data will be decoded as an instruction rather than as data. This problem can be overcome by means of the interactive disassembler provided by the SYSEDIT keyword.

Related Keywords

OPCODE$, PEEK$, SYSEDIT

NEXTOP$ (continued)

Authors

Pierre David, Jean-Jacques Dhénin and Janick Taillandier.

NLOOP

NLOOP (Number on the LOOP) returns the number of devices on the HP-IL loop.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	○ Device Operation

```
NLOOP
NLOOP ( loop number )
```

Examples

`A=NLOOP`	Stores in A the number of devices on loop number 1.

```
10 RESTORE IO
20 FOR I=1 TO NLOOP
30   DISP I;DEVID$(I)
40 NEXT I
```

Displays the number and name of all devices in the HP-IL loop.

Input Parameter

Item	Description	Restrictions
loop number	Numeric expression rounded to an integer. Default : 1	1 through 3.

Operation

NLOOP returns the number of devices on the specified loop. Multiple loops are available through the dual HP-IL adapter HP-82402.

If NLOOP is used in *extended addressing* scheme (flag -22 set), a number *sseepp* is returned, where :

ss is the answer to message AES,
ee is the answer to message AEP, and
pp is the answer to message AAD.

Note : NLOOP returns the number of devices expected to be on the specified loop. Since this information is kept in memory, the HP-71 has no need to send a message on the loop. Use RESTORE IO to update the loop information

After RESET HPIL, NLOOP returns 0.

Note : in device mode, loop data cannot be updated, therefore NLOOP cannot return valid result.

NLOOP (continued)

References

JPC 30 (page 50) first version by Jean-François Garnier.

JPC 37 (page 33) second version by Jean-François Garnier.

HP-IL Module Internal Design Specification, chapter 5.9.1.2.

The HP-IL System : An Introductory Guide to the Hewlett-Packard Interface Loop, by Gery Kane, Steve Harper and David Ushijima, (Mc Graw-Hill).

The HP-IL Interface Specification part number HP-82166-90017. Difficult, but the most precise reference.

Related Keywords

`RESTORE IO, SEND`

Author

Jean-François Garnier

NPRIM

NPRIM (Number of PRIMe numbers) returns the number of prime numbers in an interval.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
NPRIM ( n1 , n2 )
```

Example

```
A=NPRIM(10,10000)
```

Returns 1225 in 1'30". There are 1225 prime numbers between 10 and 10000.

Input Parameters

Item	Description	Restrictions
n1, n2	Numeric expressions.	Integer numbers between $-10^{12}+1$ and $10^{12}-1$.

Operation

NPRIM returns the number of prime numbers in an interval. If *n1* or *n2* are prime, they are counted in the result.

Interrupting the function with [ATTN] :

This function can be interrupted by pressing [ATTN] twice. The HP-71 will then report the error JPC ERR:Function Interrupted.

References

JPC 35 (page 21) first version of DIVILEX by Guy Toublanc.

JPC 38 (page 18) second version by Guy Toublanc.

JPC 48 (page 23) third version of DIVILEX by Guy Toublanc.

NPRIM was previously called NPRM.

Related Keywords

FPRIM, PHI, PRIM

Author

Guy Toublanc

OPCODE$

OPCODE$ returns the mnemonic of the machine language instruction pointed to by the specified address.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
OPCODE$ ( hexadecimal address )
```

Examples

`A$=OPCODE$("0BD38")`

Stores the string `A=DAT1 7` in variable A$: this is the mnemonic of the first instructiono of the POP1S routine.

`DISP OPCODE$(ENTRY$("LOG"))`

Displays `GOSUB #0BD8D`, the first instruction of the `LOG` routine.

Input Parameter

Item	Description	Restrictions
hexadecimal address	String expression containing hexadecimal digits.	Up to 5 uppercase or lowercase digits.

Operation

The OPCODE$ function :

OPCODE$ returns the mnemonic of the assembler instruction at the the specified address.

It makes the development of a disassembler much easier. For example, the following routine is a very fast, powerful disassembler. It replaces a 4+ Kbyte Basic program.

```
100 DIM I$[8],A$[5]           ! strings definitions
110 FINPUT I$,"Keyword : ",A  ! keyword input
120 IF NOT A THEN END
130 A$=ENTRY$(I$)             ! entry-point address
140 ATTN OFF
150 WHILE KEY$#"#43"          ! while [ATTN] is no pressed
160   DISP OPCODE$(A$)        !   display the instruction
170   A$=NEXTOP$(A$)          !   next address
180 END WHILE                 ! end while
190 ATTN ON
200 BEEP
```

OPCODE$ (continued)

The mnemonics :

Mnemonics returned by OPCODE$ conform to the syntax of the HP-71 assembler. The only exception are the conditional instructions which are decoded on a single line and separated by a slash (/).

The following program allows you to reformat the disassembler output to conform with the HP-71 assembler :

```
160 O$=OPCODE$(A$)                ! the instruction
161 P=POS(O$,"/")
162 IF P THEN                     ! any slash
163   DISP O$[1,P-1]              ! yes : display the test
164   DISP O$[P+1]                !         then RTNYES or GOYES
165 ELSE                          ! no : display an unmodified
166   DISP O$                     !         mnemonic
167 END IF
```

Mnemonics cannot be more than 23 characters long. A `DIM O$[23]` on line 100 is a useful complement to other definitions.

Instructions such as P= *n*, A=DAT1 *n*, ST=1 *n* are disassembled using *decimal* values. This is consistent with the Forth Rom assembler.

Instructions using absolute addresses (branch instructions, D0=(*n*), etc.) are displayed using hexadecimal constants, with a # before the constant.

The instruction LCHEX is a special case. As the name indicates that a hexadecimal value is expected, there is no need to emphasize the fact. So there is no # to clarify the data.

References

Forth / Assembler Owner's Manual : page 55. Good introduction to HP-71 microprocessor instructions.

Internal Design Specification Volume I (Chapter 16). Complete and detailed description of the instruction set.

Internal Design Specification Volume III. The way the HP-71 uses its instructions...

Related Keywords

NEXTOP$, PEEK$, SYSEDIT

Authors

Pierre David, Jean-Jacques Dhénin and Janick Taillandier.

PAGELEN

PAGELEN (PAGE LENgth) sets the page and text lengths on the printer.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	○ Device Operation

```
PAGELEN
PAGELEN page length
PAGELEN page length , text length
```

Example

```
IF K=72 THEN PAGELEN ELSE PAGELEN K,K-6 @ PERF ON
```

Input Parameters

Item	Description	Restrictions
page length	Numeric expression rounded to an integer.	0 through 999.
text length	Numeric expression rounded to an integer. Default : 72, 66	0 through 999.

Operation

PAGELEN is used to set the page size on Hewlett-Packard printers.

The logical page size is the page length in number of lines. It is equal to the product of the line spacing (number of lines per inch) and the physical length of the page in inches.

For example, with 6 lines per inch (common default value), a 12 inches page holds 72 lines, an 11 inches page 66 lines.

The 12 inches format is more or less equivalent to A4 international standard. The 11 inches format is used in the United States.

The text area corresponds to the printable page area. This is the number of lines that can be printed before skipping to next page if *perforation skip* mode is enabled. Using a 66 lines text area with a 72 lines logical page gives a 6 lines margin divided equally between the top and bottom of the page.

The PAGELEN keyword :

PAGELEN has 3 forms :

PAGELEN (continued)

The first one has no parameter. The logical page size is set to 72 lines, the text area size to 66 lines, and perforation skip is enabled. PAGELEN without parameter is equivalent to :
`PAGELEN 72,66 @ PERF ON`

The international size is the default one. If you are using 11 inches paper, you have only to execute `PERF ON`, because printers generally use a default 11 inches size.

The second form has only one parameter : the logical page size. It must be noticed that some printers, specially the ThinkJet, set a default text area length after receiving the logical page size. Consult your printer reference manual.

The third form uses two parameters.

Note : only the first form enables perforation skip. The other ones don't. You can enable it using the statement `PERF ON`.

Escape sequences sent to the printer :

```
PAGELEN:ESC & l 72 p 66 f 1 L
PAGELEN x:ESC & l x P
PAGELEN x, y:ESC & l x P ESC & l y F
```

References

JPC 26 (page 39) first version by Pierre David.

JPC 40 (page 16) second version by Pierre David.

Consult also your printer reference manual.

PAGELEN was previously called PL.

Related Keywords

PERF, PFF

Author

Pierre David

PAINT

PAINT turns on a pixel on the HP-71 display and returns its value before modification.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
PAINT ( x , y )
PAINT ( state , x , y )
```

Examples

`C=PAINT(X,Y)`	Returns the state of the point with coordinates X and Y into C.

`10 FOR X=1 TO 132`
`20   A=PAINT(1,X,5)`
`30 NEXT X`
	Draws an horizontal line on the LCD display.

Input Parameters

Item	Description	Restrictions
state , x, y	Numeric expression rounded to an integer.	None.

Operation

PAINT is used for graphic applications using the HP-71 internal display. This function performs 2 actions :

- changes the state of a dot on the screen. If *state* is zero, the point with coordinates *(x,y)* is turned off. Otherwise, it is turned on.

- returns the old state of the point. This is the value returned by PAINT.

Coordinates origin is in the upper left corner of the display. The lower right corner has coordinates $x = 131$ and $y = 7$.

To draw a line on the display, you can do :

```
10 FOR X=0 TO 131
20   C=PAINT(1,X,5)
30 NEXT X
```

References

JPC 19 (page 25) Forth word to invert the display by Jean-Pierre Bondu.

JPC 24 (page 37) first version by Jean-Jacques Moreau.

JPC 25 (page 59) Forth word returning the address of a graphic column by Jean-Pierre Bondu.

To be published : second version by Pierre David and Janick Taillandier.

Internal Design Specification Volume I, chapter 3.2.1.

Related Keywords

`GDISP`, `GDISP$`, `INVERSE`

Authors

Pierre David, Jean-Jacques Moreau and Janick Taillandier.

PARPOLL

PARPOLL (Parallel PARPOLL) returns the result of an HP-IL loop parallel poll.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	○ Device Operation

```
PARPOLL
PARPOLL ( loop number )
```

Example

`IF PARPOLL(1) THEN GOSUB 'INTERPT'` Executes 'INTERPT' if an enabled peripheral requires service.

Input Parameter

Item	Description	Restrictions
loop number	Numeric expression rounded to an integer. Default : 1	1 through 3.

Operation

Parallel poll :

Parallel polls provide the most efficient way of checking the status of two or more peripherals. A parallel poll allows up to seven devices to notify the controller (HP-71) that they require service.

Before you can parallel poll a peripheral, your controller must enable the parallel poll mode with PPE *n* (Parallel Poll Enable) frames. The parameter *n* (0 to 7) determines the response of the target peripheral when it receives a subsequent `IDY 00` frame. This is the initialization step. This allows up to seven peripherals to respond with service request simultaneously.

PARPOLL sends an `IDY 00` frame around the loop. If an enabled peripheral requires service, it will set bit *n* in the data part of the `IDY` frame, as well as the service request bit. The `IDY 00` thus changes into an `ISR` *m* frame, with $m = 2^n$.

Some peripherals does not have the capability to answer parallel polls. Consult your peripheral manual for more informations.

Using PARPOLL :

Suppose, for example, we want to enable peripherals at address 3 to set bit 4 and peripheral at address 7 to set bit 2, when they request service (initialization routine) :

PARPOLL (continued)

```
100 SEND UNL UNT    ! Unconfigure the loop
110 SEND LISTEN 3   ! Makes the peripheral at address 3 a listener
120 SEND CMD 128+4  ! PPE 4
130 SEND UNL        ! Unlistens peripheral at address 3
140 SEND LISTEN 7   ! Makes the peripheral at address 7 a listener
150 SEND CMD 128+2  ! PPE 2
160 SEND UNL UNT    ! End of sequence
```

After this initialization, the program can use parallel polls :

```
500 P=PARPOLL
510 IF BIT(P,4) THEN GOSUB 'DEV3'
520 IF BIT(P,2) THEN GOSUB 'DEV7'
```

With this code, if either device at loop addresses 3 or 7 request service, the program will branch to the respective 'DEV3 or 'DEV7' subroutines.

Interrupting the function with [ATTN] :

This function can be interrupted by pressing the [ATTN] key twice. This causes the message HPIL ERR:Aborted. It may then be necessary to execute RESTORE IO to reactivate the HP-IL system.

References

JPC 30 (page 50) first version by Jean-François Garnier.

JPC 37 (page 33) second version by Jean-François Garnier.

The HP-IL System : An Introductory Guide to the Hewlett-Packard Interface Loop, by Gerry Kane, Steve Harper, and David Ushijima (McGraw Hill).

The HP-IL Interface Specification, part number HP-82166-90017. Difficult, but the most precise reference.

PARPOLL was previously called PPOLL.

Related Keywords

SRQ, BIT, SEND, SPOLL

Author

Jean-François Garnier

PBLIST

PBLIST (Print Basic LIST) produces a structured listing of a Basic program on the current printer device.

■ Statement	■ Keyboard Execution
O Function	O CALC Mode
O Operator	■ IF...THEN...ELSE
	O Device Operation

```
PBLIST [ INDENT indentation ][TO target ]
PBLIST file [ INDENT indentation ][TO target ]
PBLIST file , start line [ INDENT indentation ][TO target ]
PBLIST file , start line , final line [ INDENT indentation ][TO target ]
```

Examples

`PBLIST MYSUB INDENT 3`	List program MYSUB, from the first to the last line, indenting structures by 3 spaces.
`PBLIST MYSUB,10`	List line 10 of program MYSUB, without structures indenting.
`PBLIST MYSUB,10,100 INDENT 2 TO LISTE`	List program MYSUB, from line 10 to line 100, indenting structures by 2 spaces. The result is sent to file LISTE.

Input Parameters

Item	Description	Restrictions
file	String expression or unquoted string. Default : current file.	File name with optional device specifier.
start line	Integer constant identiifying a program line. Default : First program line.	1 à 9999.
final line	Integer constant identiifying a program line. Default : Start line, if specified ; otherwise, last program line in file.	Start line through 9999.
indentation	Numeric expression rounded to an integer. Default : 0 REV X : DEFAULT = 2	0 through 255.
target	String expression or unquoted string. Default : Listing is output on current DISPLAY IS device.	File name with optional device specifier.

Operation

PBLIST is similar to DBLIST, but the output is directed to the current printer device instead of the display device.

Specifying a file whose type is different from Basic produces the error : `Invalid File Type` (error 63).

PBLIST (continued)

If you specify *start line* without specifying *end line*, only one line is listed as specified. If you specify an interval and the specified line numbers do not exist, the listing starts with the first line that actually exists after *first line*. PBLIST without lines specification lists the entire file.

If the printing device is a display device, the current DELAY setting determines how long each line will be displayed.

The current PWIDTH setting determines the width of the printed listing.

To halt a listing and display the cursor simply press [ATTN].

Note : LIST and PLIST are non-programmable. DBLIST and PBLIST overcome this limitation, insofar as Basic program files are concerned.

References

JPC 18 (page 25) first version of Basic program JPCLISTE by Pierre David and Michel Martinet.

JPC 38 (page 24) first version of PBLIST by Jean-Pierre Bondu.

To be published : second version by Pierre David and Janick Taillandier.

Related Keywords

ATTN, DBLIST, PLIST, DELAY, PWIDTH, MODE, all structured programming statements

Authors

Pierre David, Jean-Pierre Bondu and Janick Taillandier.

PCR

PCR (Print Carriage Return) moves the print head to the beginning of the line.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	○ Device Operation

```
PCR
```

Example

`PCR @ PRINT TAB(55);CHR$(124)`	Moves the print head to the beginning of the line, then prints a "|" at column 55.

Operation

Carriage return :

PCR sends a carriage return (code 13) to the peripheral specified by the `PRINTER IS` statement, this moves the print head to the beginning of the line.

The HP-71 keeps in memory the theoretical print head position. This value is used by TAB in a `PRINT` statement.

This position is reset to 0 after executing PCR. This allows using TAB more effectively.

Note : this statement is very useful when you are sending printer control codes and escape sequences to your printer, as the HP-71 may include such codes and sequences in the character count, which would result in incorrect print head information and in premature printing of the contents of the print buffer.

Codes sent to the printer :

Character code 13.

References

JPC 26 (page 39) first version by Pierre David.

JPC 40 (page 16) second version by Pierre David.

Consult the reference manual of your printer.

PCR was previously called CR.

Related Keywords

`PFF`, `PLF`, `PRINT TAB`

Author

Pierre David

PDIR
E1/6F

PDIR (Print DIRectory) prints directory of the specified device.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	○ Device Operation

```
PDIR [ TO target ]
PDIR file specifier [ TO target ]
PDIR ALL [ TO target ]
```

Examples

`PDIR :TAPE`	Prints directory of mass storage unit :TAPE.
`PDIR :PORT(0) TO LISTE`	Prints directory of port number 0 into file LISTE.
`PDIR ALL`	Prints all files in HP-71.
`PDIR ESSAI:TAPE(2) TO A$`	Prints all files after file ESSAI on mass storage unit :TAP(2) into the file whose name is specified by A$.

Input Parameters

Item	Description	Restrictions
file specifier	String expression or unquoted string. Default : :MAIN	Device specifier or file specifier with optional device specifier.
target	String expression or unquoted string. Default : Listing on DISPLAY IS device.	File specifier with optional device specifier.

Operation

The PDIR is identical to that of DDIR except the output goes to the print device instead of to the display device.

However, if an output redirection is specified (by TO), there is no differences between DDIR and PDIR.

References

To be published : first version by Jean-Jacques Dhénin.

Related Keywords

`CAT$`, `CAT`, `DBLIST`, `DDIR`, `PBLIST`

Author

Jean-Jacques Dhénin

PEEK$

PEEK$ returns the contents of a memory area specified by its address.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
PEEK$ ( hexadecimal address , number of nibbles )
```

Example

`A$=PEEK$(ADDR$("EXAMPLE"),2)`	**Stores "54" into variable A$, this is the internal representation of the file name first character.**

Input Parameters

Item	Description	Restrictions
hexadecimal address	String expression containing hexadecimal digits.	Up to five uppercase or lowercase digits.
number of nibbles	Numeric expression rounded to an integer.	0 through 524287.

Operation

PEEK$ is essentially identical to the standard PEEK$ function. This version allows you to peek into protected areas.

PEEK$ is reserved to HP-71 experts. For example, to return the current contrast setting, use :

```
HTD(PEEK$("2E3FE",1))
```

References

JPC 23 (page 37) first version by Pierre David, Laurent Istria and Michel Martinet.

Internal Design Specification volume I, and specially chapter 3.

Related Keywords

ADBUF$, ADDR$, ENTRY$, HTA$, HTD, PEEK$, POKE

Authors

Pierre David, Laurent Istria and Michel Martinet.

PERF

PERF enables or disables the *perforation skip* mode on the current printer device.

■ Statement	■ Keyboard Execution
O Function	O CALC Mode
O Operator	■ IF...THEN...ELSE
	O Device Operation

```
PERF ON
PERF OFF
```

Example

`PERF ON`	Enables the perforation skip mode.

Operation

Perforation skip mode :

Hewlett-Packard printers using fanfold paper often have a mode preventing them from printing in the perforation area.

PERF ON enables this mode. When done, printing the last line in a page advances paper to the top of next page. So, no printing occurs on the perforations between the two pages.

PERF OFF disables this mode.

Escape sequences sent to the printer :

```
PERF ON  :ESC & 1 0 L
PERF OFF:ESC & 1 1 L
```

References

JPC 26 (page 39) first version by Pierre David.

JPC 40 (page 16) second version by Pierre David.

Consult also your printer reference manual.

Related Keywords

PAGELEN, PLF, PRINT

Author

Pierre David

PFF

PFF (Print Form Feed) advances paper to the beginning of next page.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	○ Device Operation

```
PFF
```

Example

`IF K+L>=N THEN PFF`	Begins a new page if K+L> =N.

Operation

Form feed :

PFF advances the paper on the peripheral selected by the last PRINTER IS command to the top of next page. The top of page is the beginning of the text area defined by the last PAGELEN statement or by the default printer settings.

Codes sent to the printer :

carriage return (character code 13)
form feed (character code 12)

References

JPC 26 (page 39) first version by Pierre David.

JPC 40 (page 16) second version by Pierre David.

Consult the reference manual of your printer.

PFF was previously called FF.

Related Keywords

PAGELEN, PCR, PRINTER IS

Author

Pierre David

RENAMED GCD (REV X)

PGCD

PGCD computes the greatest common divisor of two or more numbers.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
PGCD ( arg1 , arg2 )
PGCD ( arg1 , arg2 , arg3 )
  :
PGCD ( arg1 , arg2 , arg3 , ...arg10 )
```

Example

`A=PGCD(385,210,715)` Stores 5 into A.

Input Parameter

Item	Description	Restrictions
arg_i	Numeric expressions.	Integer numbers between $-10^{12}+1$ and $10^{12}-1$.

Operation

PGCD returns the greatest common divisor of up to 10 numbers.

References

JPC 35 (page 21) first version of DIVILEX by Guy Toublanc.

JPC 38 (page 18) second version by Guy Toublanc.

JPC 48 (page 23) third version by Guy Toublanc.

Related Keywords

DIV, PPCM, PHI

Author

Guy Toublanc

PHI returns the number of relatively prime numbers lower than the argument.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
PHI ( argument )
```

Examples

`A=PHI(251)`	Stores 250 into A.
`A=PHI(999)`	648
`A=PHI(1)`	1

Input Parameter

Item	Description	Restrictions
argument	Numeric expression.	Integer number, non-zero, between $-10^{12}+1$ and $12^{12}-1$.

Operation

PHI (x) is the number of integers between 1 and x that are relatively prime to x ; this is the Euler indicator.

Interrupting the function with [ATTN] :

This function can be interrupted by pressing [ATTN] twice. The HP-71 will then report the error `JPC ERR:Function Interrupted`.

References

JPC 35 (page 21) first version of DIVILEX by Guy Toublanc.

JPC 38 (page 18) second version by Guy Toublanc.

JPC 48 (page 23) third version by Guy Toublanc.

Related Keywords

`PRIM`, `NPRIM`, `FPRIM`

Author

Guy Toublanc

PLF

PLF (Print Line Feed) advances the paper by the number of lines specified.

■ Statement	■ Keyboard Execution
O Function	O CALC Mode
O Operator	■ IF...THEN...ELSE
	O Device Operation

```
PLF
PLF number of lines
```

Examples

`IF S THEN PLF`	Skips a line is S is non-zero.
`PLF 10`	Skips 10 lines.

Input Parameter

Item	Description	Restrictions
number of lines	Numeric expression rounded to an integer. Default : 1	0 through 1048575

Operation

PLF advances the paper on the current PRINTER IS device.

The paper is advanced by the number of lines specified. If no parameter is given, one line is skipped.

If the paper reaches the end of the text area defined by either the last PAGELEN statement or by the default settings of the printer, with *perforation skip* mode enabled, the paper skips to the top of next page. Remaining lines, at the bottom of the page, are skipped.

Codes sent to the printer :

Carriage return (character code 13)
Line feed (character code 10) as needed.

References

JPC 26 (page 39) first version by Pierre David.

JPC 40 (page 16) second version by Pierre David.

Consult the reference manual of your printer.

PLF was previously called LF.

Related Keywords

PAGELEN, PERF, PFF, PRINT

Author

Pierre David

REMOVED FROM
REV X.

POKE

POKE writes to memory at the specified hexadecimal address.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
POKE hexadecimal address , data
```

Example

Be careful...

Input Parameters

Item	Description	Restrictions
hexadecimal address	String expression containing hexadecimal digits.	Up to five uppercase or lowercase digits.
data	String expression containing hexadecimal digits.	None.

Operation

POKE is similar to the standard POKE command, but does not check for file protection.

Warning : POKE is a keyword for experts. Careless use will corrupt memory and cause Memory Lost.

References

JPC 23 (page 37) first version by Pierre David, Laurent Istria and Michel Martinet.

Internal Design Specification, Volume I specially chapter 3.

Related Keywords

ADBUF$, ADDR$, ATH$, DTH$, PEEK$, POKE

Authors

Pierre David, Laurent Istria and Michel Martinet.

POSI

`POSI` (POSition in an Interval) returns the position in a string of the first character whose value falls within a specified range.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
POSI ( string , min )
POSI ( string , min , max )
```

Examples

`A=POSI("Valeur = 1000 F",48,57)`	Returns the position of the first digit in the string, because 48 is the code of "0" and 57 of "9".
`A=POSI(A$,"a","z")`	Returns the position of the first lowercase letter in A$.

Input Parameters

Item	Description	Restrictions
string	String expression.	None.
min	String expression or numeric expression rounded to an integer.	If a number, must be between 0 and 255.
max	String expression or numeric expression rounded to an integer. Default : 255.	If a number, must be between 0 and 255.

Operation

`POSI` returns the position in *string* of the first character that falls in the range between *min* and *max*.

These values can be specified using numeric values, between 0 and 255, or string expressions. If string expressions are used, only the first character is considered, this is similar to NUM. A null string is equivalent to 0.

If *max* is not given, the maximum is taken by default. In other words, any code greater than *min* will be taken into account.

If *min* > *max*, both values are swapped before searching.

If no character is found, 0 is returned.

For example, if you look for an uppercase letter, you can do :

`POSI(A$,65,90)` or `POSI(A$,"A","Z")` or `POSI(A$,"A",90)`

POSI (continued)

To search for a lowercase letter :

`POSI(A$,97,122)` or `POSI(A$,"a","z")` or `POSI(A$,97,"z")`

To search for an uppercase or lowercase letter :

`POSI(UPRC$(A$),"A","Z")`

References

JPC 37 (page 25) first version of `POSI` by Jean-Pierre Bondu.

To be published : second version by Pierre David and Janick Taillandier.

Related Keywords

POS, NUM

Authors

Jean-Pierre Bondu, Pierre David and Janick Taillandier.

RENAMED LCM (REV x)

PPCM

PPCM returns the smallest common multiple of all arguments.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

PPCM (arg_1 , arg_2)
PPCM (arg_1 , arg_2 , arg_3)
:
PPCM (arg_1 , arg_2 , $arg_3 \dots arg_{10}$)

Example

`A=PPCM(385,210,715)`	returns 30030, the smallest common multiple of 385, 210 and 715.

Input Parameter

Item	Description	Restrictions
arg_i	Numeric expressions.	Integer numbers between $-10^{12}+1$ and $10^{12}-1$.

Operation

PPCM returns the smallest common multiple of all the arguments arg_i.

References

JPC 35 (page 21) first version of DIVILEX by Guy Toublanc.

JPC 38 (page 18) second version by Guy Toublanc.

JPC 48 (page 23) third version by Guy Toublanc.

Related Keywords

PGCD, DIV

PPCM (continued)

Author

Guy Toublanc

PRIM

PRIM returns 0 if a number is prime, or the smallest divisor of that number.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
PRIM ( number )
PRIM ( higher part , lower part )
```

Examples

A=PRIM(999997874844~~)~~, 049) Returns 31622743 in 3"01.

A=PRIM(100071000730,021) Returns 10007 in 11"15.

Input Parameters

Item	Description	Restrictions
number	Numeric expression.	Integer between 1 and 10^{12}-1.
higher part	Numeric expression.	Integer between $-10^{12}+1$ and 10^{12}-1.
lower part	Numeric expression.	Integer between 1 and 999.

Operation

PRIM tests if a number is prime, and returns either 0 if it is, or the smallest divisor.

Numbers are limited to 15 digits precision. If the number falls in the 13 to 15 digits range, it must be entered as two parameters (see the above example) where :

number = *higher part* * 1000 + *lower part*

Interrupting the function with [ATTN] :

This function can be interrupted by pressing [ATTN] twice. The HP-71 will then report the error `JPC ERR:Function Interrupted`.

References

JPC 26 (page 37) first version of PRIM by Olivier Arbey.

JPC 35 (page 21) first version of DIVILEX by Guy Toublanc.

JPC 38 (pagc 18) second version by Guy Toublanc.

JPC 48 (page 23) third version by Guy Toublanc.

Related Keywords

`PGCD`, `NPRIM`, `FPRIM`

Authors

Olivier Arbey and Guy Toublanc.

REV X: PRIM(1)=1

VER A-E: PRIM(1)=0

RED$

RED$ trims all leading and trailing spaces from the specified string.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
RED$ ( string )
```

Example

`A$=RED$("  A b   c  ")`	Removes spaces before and after the string. Spaces bracketed by other characters remain unchanged.

Input Parameter

Item	Description	Restrictions
string	String expression.	None.

Operation

RED$ trims all the leading and trailing spaces from the specified parameter string.

Example :

```
10 DIM C$[50]
20 I=0
30 REPEAT
40   I=I+1
50   C$=CAT$(I)
60   C$=RED$(C$[1,8]) ! isolates the file name and trims spaces
70   DISP C$
80 UNTIL LEN(C$)=0    ! until no more files
```

References

JPC 21 (page 31) first version by Michel Martinet.

JPC 22 (page 35) second version by Michel Martinet and Pierre David.

JPC 27 (page 34) third version by Michel Martinet.

Related Keywords

REDUCE$

Authors

Pierre David and Michel Martinet.

REDUCE$

REDUCE$ reduces all substrings consisting of two or more spaces to a single space, and removes leading and trailing spaces.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
REDUCE$ ( string )
```

Example

`A$=REDUCE$("     P   P     C     ")`	Removes leading and trailing spaces from the string and reduces the number of spaces between words to only a single space.

Input Parameter

Item	Description	Restrictions
string	String expression.	None.

Operation

REDUCE$ removes all unnecessary spaces in a string. Unnecessary spaces are defined as :

- all leading and trailing spaces,

- all spaces between words, except the single space required for word division.

References

JPC 21 (page 34) first version of the Basic text formatter by Pierre David.

JPC 26 (page 50) second version of the Basic text formatter with assembly language functions by Pierre David and Michel Martinet.

Related Keywords

CESURE, RED$

Authors

Pierre David and Michel Martinet.

RENUMREM

RENUMREM (RENUMber REMarks) renumbers a Basic program with special handling for comment lines.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

RENUMREM
RENUMREM *new start*
RENUMREM *new start* , *increment*
RENUMREM *new start* , *increment* , *old start*
RENUMREM *new start* , *increment* , *old start* , *old end*

Examples

`RENUMREM 1000,10,1000,2000`	Renumbers the program starting from line 1000, in increments of 10, beginning with line 1000 and ending with line 2000.
`RENUMREM`	Renumbers the entire file, starting from line 10, in increments of 10.

Input Parameters

Item	Description	Restrictions
new start	Integer constant. Default : 10	1 through 9999.
increment	Integer constant. Default : 10	1 through 9999.
old start	Integer constant. Default : Start of file.	1 through 9999.
old end	Integer constant. Default : End of file.	1 through 9999.

Operation

When a program is listed using `DBLIST` or `PBLIST`, the line numbers of comment lines (REM or `!`) are not displayed. This gives the impression of unordered line numbering.

RENUMREM renumbers the current Basic program, like RENUMBER, but processes comment lines in a special way.

Comment lines numbers are renumbered in increments of 1, starting with a line number as near of the preceding line as possible. For example :

RENUMREM (continued)

```
10 PRINT TAB(18);"HP-71"
30 ! first comment line
40 ! second comment line
42 ! third comment line
45 ! fourth comment line
50 IF KEY$="" THEN 50
```

becomes, after RENUMREM `100,10`:

```
100 PRINT TAB(18);"HP-71"
101 ! first comment line
102 ! second comment line
103 ! third comment line
104 ! fourth comment line
110 IF KEY$="" THEN 110
```

Comment lines have been renumbered "near" line 100. Lines 100 and 110 are properly numbered. This program will be listed by `DBLIST` or `PBLIST` as :

```
 100 PRINT TAB(18);"HP-71"
   - first comment line
     second comment line
     third comment line
     fourth comment line
 110 IF KEY$="" THEN 110
```

This makes it easy to key-in the program without the comment lines, and the listing is far more readable.

References

JPC 18 (page 25) first version of program JPCLISTE, in Basic, by Pierre David and Michel Martinet.

JPC 38 (page 24) first version by Jean-Pierre Bondu.

Related Keywords

`DBLIST`, `PBLIST`, RENUMBER

Author

Jean-Pierre Bondu

REPEAT ... UNTIL

REPEAT ... UNTIL defines a loop which is repeated until the logical expression evaluated by UNTIL statement is true.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
REPEAT
   program segment
UNTIL logical expression
```

Examples

```
10 REPEAT
20    CALL AA(I,N)
30    I=I+1
40 UNTIL I+2>N
```

Executes subprogram "AA" until the condition is true.

```
10 DATA FILE1,FILE2,FILE3,
20 REPEAT
30    READ F$
40    DISP F$
50 UNTIL F$=''
```

Reads and display data, until there is nothing left to read (the last DATA is empty).

Input Parameters

Item	Description	Restrictions
program segment	Any number of contiguous program lines.	None.
logical expression.	Numeric expression evaluated as true if non-zero.	None.

Operation

REPEAT ... UNTIL executes *program segment* until the logical expression evaluated by the UNTIL statement becomes true (non-zero).

Execution starts with the first statement following REPEAT, and continues to the UNTIL statement, where a test is performed. If the test is false, a branch will be made to the first statement following REPEAT.

When the test is true, program execution continues with the first statement following UNTIL.

The loop is executed at least once.

The statement LEAVE can be used for early (and clean) exit from the loop.

REPEAT ... UNTIL (continued)

The program segment itself can contains other structures provided that such inner structures begin and end before the outer structure ends, otherwise the error `JPC ERR:Structure Mismatch` will be reported.

References

JPC 31 (page 38) first version by Janick Taillandier.

JPC 52 : second version by Pierre David and Janick Taillandier.

HP 9000 series 200/300 Basic 4.0.

Related Keywords

`LEAVE`, `LOOP` ... `END LOOP`, `WHILE` ... `END WHILE`

Authors

Pierre David and Janick Taillandier.

REPLACE$ replaces a substring with another in the target string.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
REPLACE$ ( string , pattern1 , pattern2 )
REPLACE$ ( string , pattern1 , pattern2 , start )
REPLACE$ ( string , pattern1 , pattern2 , wild )
```

Examples

`REPLACE$ ("A B   D   E F   "," ","")`	Removes all spaces from the specified string and returns "ABCDEF".
`REPLACE$ ("ABxCDxEF", ".x", "", ".")`	Displays the string "ACEF".
`A$=REPLACE$("X1 X2 X3","X\.","X")`	Stores the string "X X X" into variable A$.
`A$=REPLACE$("X1 X2 X3","X\.","X",3)`	Stores the string "X1 X X" into variable A$, i.e. the original string processed beginning at the third character.

Input Parameters

Item	Description	Restrictions
target string	String expression.	None.
match pattern	String expression.	None.
substitute pattern	String expression.	None.
start	Numeric expression rounded to an integer. Default : 1	0 through 1048575.
wildcard	String expression. Default : Null string	LEN(wildcard) < = 1

Operation

REPLACE$ replaces, in *target string*, all occurrences of the string *match pattern* by the string *substitute pattern*.

REPLACE$ uses special conventions similar to those of the SEARCH keyword used by the text editor EDTEXT found in the *Text Editor*, *Forth / Assembler* and *Translator Pac* modules.

REPLACE$ (continued)

These conventions include the use of special characters to allow more sophisticated operations. These characters are ., @, &, ^ and $. To switch these characters to their special meaning, they must be preceded by a backslash character \. Two consecutive backslash characters are considered as a single backslash character, not as two switches.

Character	Meaning
.	Any character (wild-card character)
@	Any number of wild-card characters
&	The text that matches *match pattern* when used in *substitute pattern*
^	Beginning of a line (must be the fist character in *match pattern*)
$	End of a line (must be the last character in *match pattern*)

The *start* parameter specifies the character in *target string* where the search and substitution begin. By default, *target string* is searched for *match pattern* from beginning to end.

If the *wild-card* option is used, the wild-card character in *match pattern* will match with any character in the *target string*.

References

JPC 23 (page 34) first version by Michel Martinet.

JPC 35 (page 28) first version of RPLC$ by Jean-Jacques Moreau.

Text Editor Owner's Manual.

REPLACE$ includes functions of RPLC$.

Related Keywords

MAP, MAP$

Authors

Michel Martinet and Jean-Jacques Moreau.

ROMAN

E1/6C

ROMAN enables the *Roman* extended character set.

■ Statement	■ Keyboard Execution
O Function	O CALC Mode
O Operator	■ IF...THEN...ELSE
	■ Device Operation

```
ROMAN ON
ROMAN OFF
```

Example

```
10 SUB ML
20    ROMAN ON
30 END SUB
```

After a (Memory Lost), enables the Roman extended character set.

Operation

The Roman character set

Hewlett-Packard printers or video interfaces allow using accentuated characters. To do this, you have to use the character set called *Roman*.

For example, to print character "é" on a ThinkJet or LaserJet, you only have to do:

```
PRINT CHR$(201)
```

The following table summarizes the *Roman* character set.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0				0	@	P	'	p				‾	â	Å	Á	Þ
1			!	1	A	Q	a	q			À		ê	î	Ã	þ
2			"	2	B	R	b	r			Â		ô	Ø	ã	
3			#	3	C	S	c	s			È	°	û	Æ	Đ	
4			$	4	D	T	d	t			Ê	Ç	á	å	đ	
5			%	5	E	U	e	u			Ë	ç	é	í	Í	
6			&	6	F	V	f	v			Î	Ñ	ó	ø	Ì	–
7			'	7	G	W	g	w			Ï	ñ	ú	æ	Ó	¼
8			(	8	H	X	h	x			´	¡	à	Ä	Ò	½
9			)	9	I	Y	i	y			`	¿	è	ì	Õ	ª
A			*	:	J	Z	j	z			ˆ	¤	ò	Ö	õ	º
B			+	;	K	[	k	{			¨	£	ù	Ü	Š	«
C			,	<	L	\	l	|			˜	¥	ä	É	š	■
D			-	=	M	]	m	}			Ù	§	ë	ï	Ú	»
E			.	>	N	^	n	~			Û	ƒ	ö	ß	Ÿ	±
F			/	?	O	_	o				₤	¢	ü	Ô	ÿ	

ROMAN (continued)

The first half of the table is the standard ASCII character set (see ASC$ function). The second one is called *Roman Extension*.

It can be noticed that, in both half, the first two columns are not used. These are control characters.

HP-71 character set

The HP-71 only knows the first half of the above table. The characters of the second half are mapped on the first one.

The above example is correct either you have executed ROMAN ON or not. In this latter case, the printer recognizes the character. Your HP-71 will display a character which doesn't looks like "é" unless you have executed ROMAN ON.

Using ROMAN ON allows you to have accentuated characters *in* the HP-71 display.

From the keyboard

In your programs, you can use accentuated characters as CHR$ (...). However, it is easier and more readable to create key definitions. For example, to produce *JPC*, we are using the following set :

```
DEF KEY 'fW', CHR$(197);      é
DEF KEY 'fE', CHR$(193);      ê
DEF KEY 'fR', CHR$(201);      è
DEF KEY 'fY', CHR$(203);      ù
DEF KEY 'fU', CHR$(195);      û
DEF KEY 'fI', CHR$(209);      î
DEF KEY 'fO', CHR$(194);      ô
DEF KEY 'f/', CHR$(92);       \
DEF KEY 'fA', CHR$(192);      â
DEF KEY 'fS', CHR$(200);      à
DEF KEY 'fD', CHR$(205);      ë
DEF KEY 'fJ', CHR$(207);      ü
DEF KEY 'fK', CHR$(221);      ï
DEF KEY 'f*', CHR$(124);      |
DEF KEY 'fC', CHR$(181);      ç
```

Note the semi-column after the definitions and the lack of quotes around CHR$ (...). So you enter the character and not the string CHR$ (...). You have only to press the [f] [W] key to have a "é" in the display.

Summary

HP printers use the *Roman* character set.

The HP-71 does not display character codes greater than 127 if you have not executed ROMAN ON.

Key redefinitions ease *Roman* characters input from the keyboard.

References

JPC 35 (page 8) first version of CHARLEX by Pierre David.

To be published : first version of ROMAN by Pierre David and Janick Taillandier.

Consult your printer reference manual.

Related Keywords

ASC$, CHR$, DEF KEY, NUM, MAP, MAP$

Authors

Pierre David and Janick Taillandier.

RREC$

RREC$ (Read RECord) reads a record from the specified mass storage device.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	○ Device Operation

```
RREC$ ( address , device specifier )
```

Example

```
DIM A$[256]
A$=RREC$(2,":TAPE")
```

Reads the third record of the medium directory.

Input Parameters

Item	Description	Restrictions
address	Numeric expression rounded to an integer or string expression containing hexadecimal digits specifying an address on the medium.	0 through medium maximum size.
device specifier	See standard HP-IL definitions.	Unquoted strings are not allowed.

Operation

The *record* (256 bytes) is the basic unit for transfers between the HP-71 and a mass storage device.

RREC$ reads a 256 bytes record from the medium.

Interrupting the function with [ATTN] :

This function can be interrupted by pressing the [ATTN] key twice. This causes the message HPIL ERR:Aborted. It may then be necessary to execute RESTORE IO to reactivate the HP-IL system.

References

JPC 45 (page 15) first version by Michel Martinet.

HP82161 Digital Cassette Drive Owner's Manual.

HP-IL Interface Owner's Manual Chapter 3 and appendix D.

Related Keywords

ENTER, WREC

Author

Michel Martinet

SELECT ... CASE ... END SELECT

The construct SELECT ... CASE ... END SELECT provides conditional execution of program segments.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
SELECT expression
  CASE match item
    program segment
  CASE match item
    program segment
     :
  [ CASE ELSE
      program segment ]
END SELECT
```

Examples

```
10 SELECT E+2
20   CASE <0
30     DISP "Positive"
40   CASE =0
50     DISP "Zero"
60   CASE ELSE
70     DISP "Other" @ BEEP
80 END SELECT
```

returns a message according to the value of E+2.

```
10 SELECT E$
20   CASE "A" TO "Z"
30     DISP "Uppercase"
40   CASE ":",";",",","."
50     DISP "Punctuation"
60 END SELECT
```

There is no CASE ELSE. The first choice is an interval type, the second one is an enumerated type.

Input Parameters

Item	Description	Restrictions
expression	Numeric or string expression.	None.
match item	See CASE statement.	Must be the same type as the SELECT expression.
program segment	Any number of contiguous line.	None.

Operation

SELECT ... END SELECT is similar to the IF ... THEN ... ELSE ... END IF construct, but allows several conditional program segments to be defined. Only one will be executed. Each segment starts after a CASE or CASE ELSE statement and ends when the next program line is a CASE, CASE ELSE or END SELECT statement.

The SELECT statement specifies an expression whose value is compared to the list of values found in each CASE statement. When a match is found, the corresponding program segment is executed. The remaining segments are skipped and execution continues with the first statement following END SELECT.

All *match item* must be of the same type, (either numeric or string) and must agree in type with the corresponding SELECT expression.

The optional CASE ELSE statement defines a program segment to be executed when the selected expression's value fails to match any *match item*.

Errors encountered in evaluating the *match items* are reported as having occurred in the corresponding SELECT statement.

Program segments may contain other loop or choice structures, provided that nesting is correct. Otherwise JPC ERR:Structure Mismatch will be reported.

References

JPC 52 : first version by Pierre David and Janick Taillandier.

HP 9000 series 200/300 Basic 4.0.

Related Keywords

IF ... THEN ... ELSE ... END IF

Authors

Pierre David and Janick Taillandier.

SHRINK

SHRINK minimizes the size of a text file in Ram, releasing memory that is not used to store text.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
SHRINK file
```

Example

```
10 COPY :TAPE TO A$
20 SHRINK A$
30 PURGE A$&":TAPE"
40 COPY A$ TO :TAPE
```

Copy the file whose name is in variable A$, shrinks it, purge it from the medium (very important), and copy it back to the medium.

Input Parameter

Item	Description	Restrictions
file	String expression or unquoted string.	File name with optional device specifier (the device may not be an external one).

Operation

The problem :

When text files are stored onto magnetic media, the file size is rounded up to a multiple of 256 bytes.

If you copy a text file from mass storage and add only one character to it, using your favourite editor, its size will grow. Then, when you copy it back to mass storage, the new copy will have increased by 256 bytes, 255 of which are unused. After you repeat this process a few times, as with frequently updated files, the file may include thousands of unused bytes. This wastes limited available memory.

It is also possible to get a text file with unused space if you specify a file size with the CREATE command or if you issue a PRINT # when the file pointer is in the middle of the file.

The unused space is to be found between the end-of-file mark and the physical file end.

The solution :

The SHRINK statement returns unused space to available memory.

The old file must be purged from the mass memory device before being copied back onto it, otherwise the old size is unchanged.

References

JPC 35 (page 35) first version by Jean-Jacques Moreau.

Related Keywords

COPY, EDTEXT, PURGE

Author

Jean-Jacques Moreau

SLEEP

SLEEP puts the HP-71 into light sleep mode.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
SLEEP
```

Example

`10 IF NOT KEYDOWN THEN SLEEP`	puts the HP-71 into light sleep mode if no key is pressed.

Operation

Under some conditions, it is necessary to respond quickly reactions to distant events. This occurs, for example, in data acquisition applications.

An HP-71 placed in the deep sleep mode by BYE or OFF needs too much time to power on. On the other hand, remaining in the run mode wastes power.

SLEEP puts the HP-71 in the light sleep mode. The display is kept on and HP-IL or keyboard interruptions can be processed quicker.

Pressing any key, or any interrupt, will wake the HP-71.

References

JPC 30 (page 50) first version by Jean-François Garnier.

JPC 37 (page 33) second version by Jean-François Garnier.

Related Keywords

BYE, KEYWAIT$, OFF, ON TIMER, ON INTR

Author

Jean-François Garnier

SPACE$

SPACE$ returns a string consisting of the specified number of space characters.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
SPACE$ ( repeat )
SPACE$ ( string , repeat )
SPACE$ ( character , repeat )
```

Examples

`A$=SPACE$(5)`	Stores a 5 spaces string into A$.

`10 DIM X$[50]`
`20 X$=SPACE$('TRIAL',10)`
	Repeats the string "TRIAL" ten times and stores it in X$.

`DISP SPACE$(65,20)`	Displays the character A (code 65) 20 times.

Input Parameters

Item	Description	Restrictions
repeat	Numeric expression rounded to an integer.	-1048575 through 1048575.
string	String expression.	None
character	Numeric expression rounded to an integer.	0 through 255.

Operation

SPACE$ returns a string composed of the specified number of characters. If this number is negative or zero, a null string is returned.

SPACE$ is useful in all kinds of text formatting applications, to set margins, align columns, center text, and more. It can also help initialize string arrays to be used with \SPACES$ allows you to repeat any string. This corresponds to the second and third forms. In the third case, the string is specified by the numeric code of the character.

So, `SPACE$("TRIAL ",3)` returns the string TRIAL TRIAL TRIAL . And `SPACE$(65,2)` returns AA, as 65 is the ASCII code of letter A.
FINPUT.\

References

JPC 21 (page 34) first version of the Basic text formatter by Pierre David.

JPC 26 (page 50) second version of the Basic text formatter with assembly language functions by Pierre David and Michel Martinet.

Related Keywords

CENTER$, FORMAT$

Authors

Pierre David and Michel Martinet.

SRQ

SRQ (Service ReQuest) sends a message on the HP-IL loop to check whether a peripheral requires service.

○ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
○ Operator	■ IF...THEN...ELSE
	○ Device Operation

```
SRQ
SRQ  ( loop number )
```

Example

`N=SRQ(2) @ IF N THEN GOSUB 'INTERPT'` If a peripheral requests service on the second loop, executes the specified subprogram.

Input Parameter

Item	Description	Restrictions
loop number	Numeric expression rounded to an integer. Default : 1	1 through 3.

Operation

Service request :

The service request is a process which allows peripheral request service from the HP-IL controller.

For example, the ThinkJet printer signals an "out-of-paper" condition using service requests.

If the controller (HP-71) sends data (DAB) frames or identification messages (IDY frames), the peripheral sets a dedicated bit in the frame, to indicate its service request.

After receiving the request, the HP-71 will typically query each peripheral, one at a time, with the SPOLL function to identify the device that is requesting service and to determine the problem.

SRQ provides a fast method to determine whether some peripheral requires service. Unlike parallel poll, it does not help identify the device that is requesting service see (PARPOLL). On the other hand, SRQ does not require an initialization routine.

The function :

SRQ sends an IDY 00 frame on the specified loop, and returns the state of the service request bit. The value returned by SRQ will be :

SRQ (continued)

- 1 if one or more peripheral request service, or
- 0 if no peripheral request service.

Interrupting the function with [ATTN] :

This function can be interrupted by pressing the [ATTN] key twice. This causes the message `HPIL ERR:Aborted`. It may then be necessary to execute `RESTORE IO` to reactivate the HP-IL system.

References

JPC 30 (page 50) first version by Jean-François Garnier.

JPC 37 (page 33) second version by Jean-François Garnier.

The HP-IL System : An Introductory Guide to the Hewlett-Packard Interface Loop, by Gerry Kane, Steve Harper, and David Ushijima (McGraw Hill).

The HP-IL Interface Specification, part number HP-82166-90017. Difficult, but the most precise reference.

Related Keywords

`PARPOLL`, `SEND`

Author

Jean-François Garnier

STACK

STACK sets the size of the command stack to the specified number of levels.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
STACK number of levels
```

Example

`STACK 15`	**Initialize the command stack to 15 levels.**

Input Parameter

Item	Description	Restrictions
number of levels	Numeric expression rounded to an integer.	1 through 16.

Operation

The command stack :

The command stack provides a convenient way to reexecute keystroke sequences without retyping the entire sequence. In the standard HP-71, this stack is limited to only 5 levels.

The STACK statement :

The STACK statement changes the size of the command stack to the specified number of levels.

A one level stack maximizes available memory but is not very useful as a typing aid. A sixteen level stack can use up to 1400 bytes of Ram. Also when you use the INPUT or LINPUT statements, the [v] key will wrap-around the top of the command stack. This is unexpected, and can confuse the user.

A 15 level stack is a good compromise, as it saves a maximum of keystroke sequences while avoiding unexpected behavior. This is the type of command to use in the ML program, automatically activated after a Memory Lost.

References

User's Library Solutions - Utilities, SETCMDST subprogram (page 3).

JPC 25 (page 57) first version by Michel Martinet.

To be published : second version by Henri Kudelski.

STACK (continued)

Related Keywords

`INIT 3`

Authors

Henri Kudelski and Michel Martinet.

STARTUP$

STARTUP$ returns the STARTUP command string.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
STARTUP$
```

Example

`A$=STARTUP$`	Stores in A$ the command string executed each time the HP-71 is turned on.

Operation

STARTUP$ returns the command string to be executed each time the HP-71 is turned on. This string cannot be more than 95 characters long.

STARTUP$ returns a null string unless you have previously stored a command string in the startup buffer with the STARTUP statement.

References

JPC 25 (page 43) first version by Jean-Jacques Moreau.

JPC 31 (page 29) second version by Jean-Jacques Moreau.

Related Keywords

ENDUP$, STARTUP

Author

Jean-Jacques Moreau

SYSEDIT

SYSEDIT (SYStem EDITor) puts the HP-71 into an interactive memory editor / disassembler mode.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

SYSEDIT [*hexadecimal address*] REV X: address is optional

Examples

SYSEDIT "0BD38" — Display the memory contents beginning at address 0BD38, i.e. at the start of POP1S.

SYSEDIT ADDR$("JPCLEX") — Display memory starting at JPCLEX file header.

Input Parameter

Item	Description	Restrictions
hexadecimal address	String expression containing hexadecimal digits. REV X: default address is wherever it was when you last used it! *	Up to 5 uppercase or lowercase digits.

*uses reserved RAM.

Operation

SYSEDIT sets the HP-71 into an interactive memory editor / disassembler mode. In this mode, the contents of memory are displayed as either hexadecimal characters or as assembler instructions or macro-instructions. The display initially looks like :

```
00000:2034EE100060F481
```

The first part is the *address*.
The second part which is separated from the first one by a ":" is the contents of memory at that address.

The following keystroke sequences allow you to use the interactive editor :

[ATTN] or **[f] [OFF]** - Exits the editor
SYSEDIT exit.

[+], **[-]**, **[*]** or **[/]** - Moves the editor window through memory
Operations on the current address, respectively : +1, -1, +16 et -16. In each case, hexadecimal display mode is turned on by default. REV X: STAYS IN DISASSEMBLE MODE!

[A] [1] to **[A] [8]** - Display memory as ASCII characters

SYSEDIT (continued)

Displays the contents of memory as NIBASC assembler instruction. The number following [A] is the number of characters you wish to display. Non displayable characters (outside the 32 to 126 range) are displayed as a dot.

[N] [1] to [N] [9] and [N] [.] [0] to [N] [.] [6] -
Display memory as hexadecimal nibbles
Displays the contents of memory as NIBHEX assembler instruction. The number following [N] is the number of hexadecimal digits you wish to be displayed. If this number is greater than 9, it must be preceded by a dot. For this example, [N] [8] displays 8 digits, but [N] [.] [6] displays 16 digits.

[C] [1] to [C] [6] - Display memory as a constant
Displays the contents of memory as CON assembler instruction. The number following [C] is the constant number of nibbles you wish displayed. If this number is less than 4, the constant value is given in decimal form, otherwise it is given in hexadecimal form(with a #).

[C] [H] [1] to [C] [H] [6] - Display memory as hexadecimal
constants
This option is the same as the previous one, but the display is set to hexadecimal mode.

[R] [1] to [R] [5] - Display relative address
Display the memory contents as a relative address (macro-operation REL). The computed address is always displayed in hexadecimal.

[H] - Hexadecimal mode (default)
Set hexadecimal mode : display memory contents as 16 hexadecimal digits.

[D] - Disassembler mode
Enter the disassembler mode : display memory contents as assembler mnemonics. See OPCODE$ for more details about the format used to display this information.

[L] - Load ASCII constant
If disassembler mode is active and if the currently disassembled instruction is a LC(*n*), with *n* even, the instruction is displayed as a LCASC mnemonic. Non displayable characters, whose code is not between 32 and 126) are displayed as a dot.

[F] - Saving disassembler output
Asks for a file name. Thereafter, each time you press [ENDLINE], the contents of the display will be appended to the file you specify. The file is created if needed, otherwise the pointer is moved to the end of the file. To exit from this mode, clear the display with[f] [-LINE] and press [ENDLINE].

[=] - Direct move
If an address is in the display, the editor will branch to the address.

[(] - Move and push address
If an address is in the display, the current address is pushed on the stack and the editor will branch to the address displayed. The stack can contain up to 7 addresses. This allows you to trace subroutine calls.

[)] - Return
If the address stack is not empty, pop the last address and return the editor to the address specified by the stack.

[ENDLINE] - Validation
If file mode is active this stores the display contents in the file, skip the current object and increment the address by the length of the object (opcode, characters, relative address, constant) being displayed. If the disassembler mode is active, displays the mnemonic of the next instruction, otherwise defaults to hexadecimal display mode.

[Z] - Address editing

Interactive editing of the address. To store the modifications, press [ENDLINE]. To stop without making changes, press [ATTN].

[f] [Z] or [M] - Memory editing
Interactive editing of the contents of memory displayed in hexadecimal characters. To store any changes, press [ENDLINE]. To exit without making any change, press [ATTN].

References

JPC 22 (page 31) first version of a Basic disassembler by Michel Martinet.

Not published : first version of SYSEDIT in Basic by Pierre David.

Not published : SYSEDIT by Pierre David and Janick Taillandier.

Forth / Assembler Owner's Manual : page 55. Good introduction to HP-71 microprocessor instructions.

Internal Design Specification Volume I (Chapter 16). Complete and detailed description of the instruction set.

Internal Design Specification Volume III. The way the HP-71 uses its instructions...

Related Keywords

ADDR$, OPCODE$, PEEK$, POKE, ENTRY$

Authors

Pierre David and Janick Taillandier.

TOKEN

TOKEN returns the Lex Id and token for the specified keyword.

○ Statement	■ Keyboard Execution
■ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
TOKEN ( keyword )
TOKEN ( keyword , sequence )
```

Example

`A=TOKEN("TOKEN",1)`	Stores 225073 into variable A, this is the resource allocation for the TOKEN keyword.

Input Parameters

Item	Description	Restrictions
keyword	String expression.	The keyword must exist.
sequence	Numeric expression rounded to an integer. Default : 1	The keyword must exist.

Operation

TOKEN returns the resource allocation for the specified keyword. This is the number used after XWORD or XFN when the Lex file containing the function has been removed.

TOKEN returns the internal code as *iittt*, where *ii* is the Lex identifier (Id) followed by the token allocation (*ttt*). For example, TOKEN itself has the token 73 in the Lex 225. The result is 225*1000+73, or 225073.

If there is more than one keyword in your HP-71 with the same name, *sequence* allows you to identify the duplicates. The default *sequence* is 1.

If the keyword does not exist or if *sequence* is greater than the number of duplicate for example, `ERR:Invalid Arg` will be reported.

For keywords of more than 8 characters, special processing is required from the system. For example, keywords like `UNDERLINE` or `RANDOMIZE` are recognized as `UNDERLIN` or `RANDOMIZ`. The final "E" is processed by the keyword itself. Therefore, TOKEN handles only the first eight characters of a keyword. Any additional characters will be ignored. Thus, TOKEN recognizes `UNDERLIN` and does not take care of the "E". Similarly, `TOKEN("RANDOMIZE")` and `TOKEN("RANDOMIZ----")` ignore extra characters and return the same number.

TOKEN (continued)

TOKEN will identify the longest keyword whose name is part of the parameter string, starting with the first character. For example, `TOKEN("MEMORY")` returns the resource data of the MEM keyword.

TOKEN recognizes all statements and functions but also all valid Basic syntax elements. So, `TOKEN(",")` is correct as well as `TOKEN("3")`, etc.

TOKEN provides a convenient method for identifying potential keyword conflicts.

References

JPC 31 (page 22) first version by Jean-Jacques Moreau.

Forth/Assembler Rom Owner's Manual (page 63).

Internal Design Specification, Volume I.

Related Keywords

`ENTRY$`

Author

Jean-Jacques Moreau

UNDERLINE

UNDERLINE enables or disables underline mode on the printer.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	○ Device Operation

```
UNDERLINE ON
UNDERLINE OFF
```

Examples

`UNDERLINE ON @ PRINT "HP-71"`	Prints the string HP-71.
`UNDERLINE OFF @ PRINT "HP-71"`	Prints the string HP-71 without underlining.

Operation

UNDERLINE ON enables the underline mode on the current PRINTER IS device.

UNDERLINE OFF returns the printer device to normal mode.

Note : the UNDERLINE ON / OFF commands are intended for printers that implement the HP *Printer Control Language* such as the ThinkJet and LaserJet printers. It may not work as expected with other types of printers.

Escape sequences sent to the printer :

```
UNDERLINE ON :ESC & d D
UNDERLINE OFF:ESC & d @
```

References

JPC 26 (page 39) first version by Pierre David.

JPC 40 (page 16) second version by Pierre David.

Also consult your printer reference manual.

Related Keywords

BELL, BOLD, MODE, PAGELEN, PCR, PFF, PLF, PRINT, PRINTER IS, WRAP

Author

Pierre David

VARSWAP

VARSWAP swaps the contents of two variables or array elements.

■ Statement	■ Keyboard Execution
O Function	O CALC Mode
O Operator	■ IF...THEN...ELSE
	■ Device Operation

```
VARSWAP variable1 , variable2
```

Examples

`VARSWAP A,B`	Swaps the contents of numeric variables A and B.
`VARSWAP A$,B$`	Swaps the contents of string variables A and B.

Input Parameters

Item	Description	Restrictions
variable1, variable2	Name of numeric or string variables.	Variable types must be compatible.

Operation

VARSWAP swaps the contents of two variables provided that they are of the same type and dimensions.

Warning !

VARSWAP does not work if *variable2* does not exist and must be created by VARSWAP.

This may cause a memory lost.

NB: VARSWAP A,B
sets RES = B.

References

JPC 31 (page 50) first version by Jean-Jacques Moreau.

To be published : second version by Pierre David and Janick Taillandier.

VARSWAP was previously called SWAP.

Related Keywords

LET

Authors

Pierre David, Jean-Jacques Moreau and Janick Taillandier.

WHILE ... END WHILE

WHILE ... END WHILE defines a loop which is executed as long as the logical expression in the WHILE is true.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	■ Device Operation

```
WHILE logical expression
   program segment
END WHILE
```

Example

```
10 WHILE I+2<=N
20   CALL AA(I,N)
30   I=I+1
40 END WHILE
```

Repeats the AA subprogram as long as the condition is true.

Input Parameters

Item	Description	Restrictions
logical expression	Numeric expression evaluated as true if non-zero.	None.
program segment	Any number of contiguous lines.	None.

Operation

The WHILE ... END WHILE construct allows conditional execution of a program segment. If the condition is true, the program segment between the WHILE and END WHILE is executed and a branch is made back to the WHILE statement.

The program segment will be repeated until the test becomes false. When this occurs, the program segment will be skipped and execution continues with the first statement after END WHILE.

The program segment may never be executed if the expression is evaluated as false the first time.

As for loop structures LOOP ... END LOOP or REPEAT ... UNTIL, the statement LEAVE allows early exit from a WHILE ... END WHILE structure.

The program segment itself can contain other structures provided that the inner structure begins and ends before the outer construct ends, otherwise the error JPC ERR:Structure Mismatch will be reported.

WHILE ... END WHILE (continued)

References

JPC 31 (page 38) first version by Janick Taillandier.

JPC 52 : second version by Pierre David and Janick Taillandier.

HP 9000 series 200/300 Basic 4.0.

Related Keywords

`LEAVE`, `LOOP ... END LOOP`, `REPEAT ... UNTIL`

Authors

Pierre David and Janick Taillandier.

WRAP

WRAP enables or disable the printer wrap-around mode.

■ Statement	■ Keyboard Execution
○ Function	○ CALC Mode
○ Operator	■ IF...THEN...ELSE
	○ Device Operation

```
WRAP ON
WRAP OFF
```

Example

`WRAP ON @ PBLIST`	Prints the current program. A carriage return / line feed is executed for lines longer than the current line length.

Operation

The wrap-around mode is used when the HP-71 must print lines longer than the printer current line length. Long lines are broken-up and printed on several lines.

The HP-71 has already a similar capability with the statement PWIDTH. However, WRAP mode is handled by the printer and eases the burden on your HP-71. Also, the HP-71 includes the escape sequences that PRINT sends to the printer to compute the line length, which may induce errors. Using WRAP insures more exact results and improved system performance.

Escape sequences sent to the printer :

```
WRAP ON  :ESC & s 0 C
WRAP OFF:ESC & s 1 C
```

References

JPC 26 (page 39) first version by Pierre David.

JPC 40 (page 16) second version by Pierre David.

Also consult your printer reference manual.

Related Keywords

BOLD, ENDLINE, ESC$, PRINT, PRINTER IS, UNDERLINE

Author

Pierre David

WREC

WREC (Write RECord) writes a 256 bytes string to the specified sector of selected mass memory device.

■ Statement	■ Keyboard Execution
O Function	O CALC Mode
O Operator	■ IF...THEN...ELSE
	O Device Operation

```
WREC sector , address , device specifier
```

Example

`WREC A$,1,:TAPE`	Writes the string A$ (256 characters) to sector number 1 of medium specified by :TAPE.

Input Parameters

Item	Description	Restrictions
sector	String expression.	The length must be exactly 256 bytes.
address	Numeric expression rounded to an integer or string expression containing hexadecimal digits specifying an address on the medium.	0 through medium maximum size.
device specifier	See standard HP-IL definition.	Unquoted strings are ~~not~~ allowed.

Operation

WREC writes a 256 bytes string (the sector or record) to the *device specifier.*

The address may be either a sector number in decimal, or a string expression containing the hexadecimal equivalent.

Interrupting the function with [ATTN] :

This function can be interrupted by pressing the [ATTN] key twice. This causes the message `HPIL ERR:Aborted`. It may then be necessary to execute `RESTORE IO` to reactivate the HP-IL system.

References

JPC 45 (page 15) first version by Michel Martinet.

HP82161 Digital Cassette Drive Owner's Manual.

HP-IL Interface Owner's Manual Chapter 3 and appendix D.

Related Keywords

`RREC$, OUTPUT`

Author

Michel Martinet

OTHER JPC ROM FEATURES

The other features of JPC Rom cannot be accessed through keywords. They are available through *polls*.

These features are not called and cannot be adjusted. They are available when JPC Rom is in memory, without any further intervention.

The following pages describe these features.

REV X ADDITIONS:

1. TEXT FILES CAN BE LISTED.
2. PRINTER IS NULL no longer crashes.

The VER$ function

VER$ returns a string indicating the version of JPC Rom present in the HP-71 memory or in a plug-in Rom.

Operation

VER$ returns a string containing JPC Rom version.

The version of JPC Rom that corresponds with this manual is : JPC:D. (REV X REPORTS: "JPC:Ex")

References

HP-71 Reference Manual

JPC 23 (page 33) poll introduction by Laurent Istria.

Internal Design Specification Volume I, chapter 8.4 and page 17.14.

Related Keywords

VER$

Authors

Pierre David and Janick Taillandier.

Assembler tabs

Assembler source files are much more readable when various fields are aligned.

Operation

About tabs :

Although the assembler processes files in *free format*, they are much more readable when aligned.

Tabs are available by pressing the [SPC] key in text editing mode. Each time you press the [SPC] key, the cursor skips to the next tab until the last tab at column 25.

Tabs are set to columns 9, 16 and 25. A «*» in the line, specifying a comment, disables all tabs.

Using tabs :

The key [SPC] moves the cursor to the next tab when :

- tab mode is enabled,
- the HP-71 is in Text Editor mode,
- when there is no star character (*) in the line.

The tab mode is enabled or disabled by pressing on [CALC] in Text Editor mode.

References

JPC 30 (page 42) third version by par Stéphane Barizien, Pierre David and Michel Martinet.

Forth / Assembler ROM Owner's manual, page 46.

Related Keywords

EDTEXT

Authors

Stéphane Barizien, Pierre David and Michel Martinet.

Editing in CALC mode

In CALC mode, access to backspace key is unconvenient. JPC Rom enables the `[<]` key to correct inputs.

Operation

CALC mode :

CALC mode is very convenient for calculations. However, editing inputs is not very easy because you have to press two keys (`[g]` and `[<-]`) to correct the last input character.

With JPC Rom in your HP-71, you have only to press `[<-]` to correct mistakes.

References

JPC 27 (page 26) first version by Pierre David.

Related Keywords

CALC mode

Author

Pierre David

Cursor position

Without video interface for a display device, it is sometimes difficult to know the current cursor position in the line.

Operation

In USER mode, press [f] [VIEW] to display the current cursor position. This display is maintained while key is held down.

Cursor position will be a number between 1 and 96.

This is available only in USER mode to allow using [VIEW] to display key assignments.

This feature is available in most operating modes, to include text editing modes, in Forth and even during `INPUT`, `LINPUT`, `FINPUT` or KA execution.

References

JPC 24 (page 33) first version by Pierre David.

Related Keywords

`DEF KEY`, `EDTEXT`, `FORTH`, `FINPUT`, `INPUT`, `KA`, `LINPUT`

Author

Pierre David

Auto-repeat speed up

JPC Rom in your HP-71 speeds-up the keyboard auto-repeat feature.

Operation

As soon as JPC Rom is in your machine you can notice that the keyboard auto-repeat response is significantly faster.

This is a permanent feature. It is available in Text Editor mode, in Forth and even while executing `INPUT`, `LINPUT`, `FINPUT` or KA. The auto-repeat is not speeded-up during catalog operations.

References

JPC 26 (page 29) first version by Jean-Jacques Moreau.

To be published : second version by Pierre David and Janick Taillandier.

Related Keywords

`EDTEXT`, `FORTH`, `FINPUT`, `INPUT`, `LINPUT`

Author

Jean-Jacques Moreau

Initialization after a Memory Lost.

After a memory reset, the HP-71 will try to execute a subprogram called ML.

Operation

This is very useful to reset all sorts of parameters after a Memory Lost. For example, date and time, delay, contrast, etc. Here is a sample subprogram :

```
100 SUB ML
110   DIM D$
120   FINPUT D$,"Time : Hr:Mn:Sc","7P2UP2UP2UP",A
130   SETTIME D$[1,2]&":"&D$[3,4]&":"&D$[5]
120   FINPUT D$,"Date : Dy/Mo/Yr","7P2UP2UP2UP",A
130   SETDATE D$[5]&"/"&D$[3,4]&"/"&D$[1,2]
140   DELAY 0,0
150   USER ON
160   STACK 15
170   LC ON
180   WIDTH 80
190   PWIDTH INF
200   DEF KEY "#46","RUN ";
210 END SUB
```

Of course, this program must be in independent Ram, Rom or Eprom module, otherwise it would be cleared during the memory reset.

Warning : this subprogram must not contain configuration statements such as : LEX ON/OFF, COPY of Lex files into Ram , etc.

References

JPC 31 (page 24) first version by Jean-Jacques Moreau.

NOTE: REV X CONTAINS NO DEFAULT ML PROGRAM.

Related Keywords

CALL, SUB

Author

Jean-Jacques Moreau

Filetype translation

JPC Rom recognized new file types and displays their name during a CAT or DDIR.

Operation

When JPC Rom is plugged in your HP-71 it decodes the file type of non standard files during a CAT, CAT$, DDIR or PDIR.

This allows you to easily recognize or locate files from other computers (HP-41 or HP-75, for example) when they are stored on a mass storage device.

The following filetypes are recognized :

HP-71 files

GRAPH	GRAPHIC modules pictures
FORTH	Forth or Translator Forthram files
ROM	Image of Independent Ram (ROMCOPY LEX)
OBJ	Object files (Development Module, not yet available)
SYM	Symbol tables (Development Module, not yet available)

HP-41 files

41:WA	HP-41 Ram backup (WRTA)
41:KE	Key assignments backup (function WRTK)
41:ST	Status backup (function WRTS)
41:PR	HP-41 Program
41:ML	Microcode programs for MLDL (Eramco)
41:XM	Extended memory backup (function WRTXM in EXT IL Rom)
41:CA	Calculator (function WRTCAL in EXt IL Rom)

HP-75 files

75:T	Text files
75:A	Alarm fiels (APPT)
75:B	Basic programs
75:L	Lex files
75:W	Visicalc spreadsheet
75:G	General purpose file (I/O Rom)
75:R	PMS Rom

References

To be published : first version by Jan Buitenhuis and Janick Taillandier.

Related Keywords

CAT, CAT$, DDIR, PDIR

Authors

Jan Buitenhuis and Janick Taillandier.

HP-IL messages

The error messages of the HP-IL module are not always very clear. JPC Rom redefines them to get more precise diagnostics.

Operation

As the HP-IL module size is exactly 16384 bytes, there is not even a free nibble in the ROM. It is easy to understand why messages are often identical. Hewlett-Packard was obliged to ignore some messages to fit the HP-IL Lex into a 16 Kbytes module.

JPC Rom redefines these messages to clarify there meanings.

Here is the message list, giving the HP-IL module message first and the corresponding JPC Rom message seconds :

0 : *non existent*
`0:HPIL`

Message 0, used by the system to display errors, for example :
`HPIL ERR:Blank Medium`

`1:ASSIGN IO Needed`
`1:ASSIGN IO Needed`

Attempted to execute a `LIST IO` without having executed `ASSIGN IO`. Execute `ASSIGN IO`.

`3:Excess Chars`
`3:Excess Chars`

The HP-71 found more characters in the command than expected. Check syntax.

`4:Missing Parm`
`4:Missing Parm`

A parameter required by the statement is missing. Check syntax.

`5:Invalid Parm`
`5:Invalid Parm`

A parameter used in the statement is not legal. Check parameters.

`6:Invalid Expr`
`6:Invalid Expr`

The expression cannot be evaluated due to invalid data type (such as a string variable instead of a numeric one). Check the expression.

`7:Syntax`
`7:Syntax`

HP-IL messages (continued)

The HP-71 does not recognize the statement. Check keyword spelling and required parameters.

16:File Protect
16:File Protect

The file is secure or private, you cannot execute this operation. If the file is secured, execute UNSECURE.

17:End Of Medium
17:End of Medium

The file is too big for available space on the medium ; medium is full ; drive error condition. Check medium ; recreate file ; pack medium ; use another medium.

Warning : whenever you have a disk drive error, there is a strong possibility that you have a low battery condition. Make sure that the battery is adequately charged before taking other corrective action ! Be sure to back-up important files before attempting a PACK operation. PACK is the most common cause for crashed mass storage media.

18:Invalid Medium
18:Disk Drive Error

Mass storage drive motor is stalled. Check for jammed medium.

19:Invalid Medium
19:Not LIF Format

The medium is not initialized to proper format. Execute INITIALIZE statement.

20:No Medium
20:No Medium

No medium detected in the mass storage device. Check that drive door is closed ; insert medium.

21: *non existent*
21:Low Battery

The mass storage has not enough power. Check or change batteries. This will not work properly with HP9114A disk drive, due to a system bug in the drive.

22:File Not Found
22:File Not Found

The specified file was not found ; the specified file name differs from directory entry. Check directory and file name.

23:Invalid Medium
23:New Medium

Opened and closed door of mass storage device during file operation or medium access. Restart the operation.

24:`Invalid Medium`
24:`Blank Medium`

Medium not initialized. Execute the `INITIALIZE` statement.

25:`Invalid Medium`
25:`Wrong dir # records`

Record number in directory does not match record number on medium. Retry. If it fails again, initialize medium, using `INITIALIZE` statement and recreate file system.

Caution : this message may indicate a low battery condition.

26:`Invalid Medium`
26:`Checksum`

Data checksum error detected. Initialize medium ; recreate file.

28:`Size of File`
28:`Size of File`

File too big to copy to or from the mass storage device. Add a memory module to the HP-71 or use another medium.

29: *non existent*
29:`Write Protected`

Disk drive error. The medium in specified drive is write protected.

30:`File Exists`
30:`File Exists`

The file name specified in a `CREATE` or as the destination of a `COPY` statement already exists. Purge old file or use another name.

31:`Directory Full`
31:`Directory Full`

The directory on the medium is full. Purge unwanted files and pack directory or medium.

32:`Device Not Found`
32:`Device Not Found`

The requested device does not exist on the loop. Check device specifier, system organization and execute `RESTORE IO`.

HP-IL messages (continued)

```
34:Device Not Ready
34:Device Not Ready
```

A device did not respond as expected. Check device specifier ; check device ; execute RESTORE IO.

```
35:Loop Broken
35:Loop Broken
```

The loop is not complete. Check connections and make sure that devices are turned on.

```
36:Message Error
36:Too Many Frames
```

The HP-71 received too many messages. Restart the operation.

```
37:Message Error
37:Frames Lost
```

Message lost due to slow retransmission. Restart operation.

```
38:Message Error
38:Frames Altered
```

Message altered during transmission. Restart operation.

```
39:Unexpected Message
39:Unexpected Message
```

HP-IL protocol violation occurered (more than one talker was active in loop at the same time).

```
40:Message Error
40:Too Many Frames
```

The HP-71 received too many messages. Restart operation.

```
41:Invalid Mode
41:Invalid Mode
```

Attempted to execute a controller statement while the HP-71 was acting as a device. Check the mode (controller or device) required by the statement.

```
42:Loop Broken
42:Message Altered
```

A partial message was received by the HP-71 due to a transmission error. Restart operation.

43:Loop Broken
43:Loop Timeout

A message took longer than the STANDBY timeout period to go around the loop. Clear listeners ; restart operation.

44:System Error
44:Bad Addresses

Device addresses probably invalid (if flag -24 set). Clear flag -24 or assign new addresses (execute RESTORE IO).

Internal error related to I/O channels. Execute RUN and restart operation ; execute INIT:1 ; execute INIT:3. If error persists the HP-71 requires repair service.

45:Self-test failed
45:Self Test Failed

The HP-IL interface failed internal self-test. Repeat self-test by executing RESET HPIL. If the error persists, the interface needs repair service.

47:Device Type
47:Device Type

The specified device is not a legal type for the statement. Check device type requirements.

52:Aborted
52:Aborted

Pressed [ATTN] twice to interrupt operation. Execute RESTORE IO ; if necessary, execute RESET HPIL, then RESTORE IO. Check HP-IL connections ; check that peripherals are turned on.

53:Invalid Device Spec
53:Invalid Device Spec

The device specifier is not valid for the statement. Check device specifier.

54:Data Type
54:Data Type

Specified the wrong type of variable (numeric or string). Change argument to the proper variable type.

56:Invalid Arg
56:Invalid Arg

An argument is out of the allowable range. Check argument value.

Directory entry (start record or length) received during mass storage operation is invalid. Re-store file.

HP-IL messages (continued)

```
57:No Loop
57:No Loop
```

Interface is not installed, plug-in HP-IL module. Check system configuration.

```
59:Insufficient Memory
59:Insufficient Memory
```

Not enough Ram to perform required operation. Add memory ; delete files or key assignments ; reallocate internal memory.

```
60:RESTORE IO Needed
60:RESTORE IO Needed
```

Attempted an I/O operation after executing `OFF IO`. Execute `RESTORE IO`.

References

JPC 37 (page 27) first version by Michel Martinet.

To be published : second version by Michel Martinet.

HP-IL interface owner's manual, Appendix E.

Related Keywords

MSG$, All HP-IL keywords

Author

Michel Martinet