The HP 48 Handbook

James Donnelly

The HP 48 Handbook

James Donnelly

Copyright © James Donnelly 1990

All rights reserved. No part of this book may be reproduced, transmitted, or stored in a retrieval system in any form or by any process, electronic, mechanical, photocopying or means yet to be invented, without specific prior written permission of the author.

First Edition

ISBN 1-879828-00-6

First Printing, May 1990
Second Printing, September 1990
Third Printing, January 1991

Armstrong Publishing Company
3135 NW Ashwood Drive
Corvallis, OR 97330 USA

Acknowledgements

Special thanks go to Eric L. Vogel for his many thoughtful comments and hours of hard work. Jake Schwartz contributed the PARENT and STKV programs. Alonzo Gariepy contributed the QSORT program, ideas, and encouragement. Ron Brooks, Sharon Butterfield, Tom Diamond, Grant Garner, Wlodek A.C. Mier-Jedrzejowicz, Scott Rohrer, Bill Wickes, Bob Worsley, and Dennis York contributed advice, ideas, support, and encouragement.

The cover photograph, taken by Peter Krupp, was provided by the Hewlett-Packard Company.

To Russell and Marian Donnelly

Contents

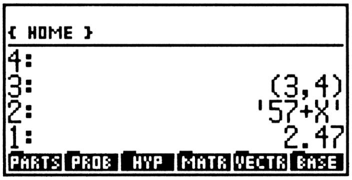
Introduction

The HP 48 Handbook is designed with the programmer in mind – a concise combination of system descriptions and detailed reference information. *The HP 48 Handbook* is not intended to be a replacement for the Owner's Manuals – which cover the interactive applications and calculus subjects not treated herein.

Organization. The first chapters cover the organization of the system, object manipulation, and how programs work. The next chapter discusses the HP Solve Equation Library application card, with both operation and reference information. The remaining chapters provide reference tables for flags, messages, units, and so on.

The Subject Index lists the commands by subject areas to provide another way to rapidly find the right command for a particular application. The Command Reference contains the complete set of stack diagrams for every command in the HP 48.

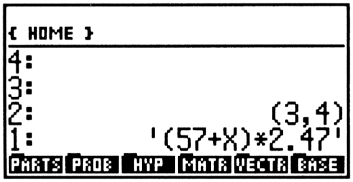
Fundamental Concepts. The HP 48 world revolves around the *stack*, which is implemented as a dynamically allocated last–in–first–out (LIFO) structure which can hold any number objects of different sizes and types (see *Objects, Names, and Constants*). All commands take their (zero or more) arguments from the stack and return any results to the stack. For instance, consider the following display:



```
{ HOME }
4:
3:                  (3,4)
2:                 '57+X'
1:                   2.47
```

Level 1 contains the number 2.47, level 2 the algebraic expression '57+X', and level 3 the complex number (3,4).

Now execute the multiply function. While multiply is executing, the arguments are removed from levels 1 and 2, leaving (3,4) in level 2. When the multiplication is complete the result is returned to the top of the stack:



```
{ HOME }
4:
3:
2:                (3,4)
1:      '(57+X)*2.47'
```

Many commands are type-sensitive, that is, they perform different operations for different types of input parameters. For the complete descriptions for each command, see *Command Reference*.

Example Programs. There are several example programs and program fragments in this book. Each complete program is named and printed with a size and checksum.

All characters in the programs are case-sensitive. The names of commands are always uppercase. By convention, the names of global variables are uppercase, and of local variables are lowercase.

While the command line entry of a program may be free form, with the [↵] keystroke being valid between words, graphics objects must be entered exactly as shown, with no extra breaks in the command line when entering the data.

If you enter a program into the HP 48, use the BYTES function to make sure the program in the calculator matches the version in the book. For instance, the program `« DROP SWAP »` is 15 bytes long and has the checksum #5197h. The sizes for named programs include the size of the program name.

Objects, Names, and Constants

Object is a general term for anything that can be put on the stack or stored in a variable. Any object may be described in terms of its *type* and *value*. For instance the number 247 has type "real number" with value 247.

Objects may be classified into several broad categories:

- A *data object* contains information, such as a number or a sequence of characters. Real numbers, complex numbers, binary integers, arrays, and strings are examples of data objects.
- A *procedure object* is a collection of objects that perform a task in order. Programs and algebraic expressions are procedure objects, and may be evaluated, placed on the stack or stored in variables just like any other object.
- A *name object* permits an object to be referenced by name.
  - □ *Global names* refer to corresponding variables that are available at any time. By convention, global variable names are written in uppercase (A).
  - □ *Local names* refer to corresponding local variables that exist only with the scope of the executing program that defines them. By convention, local variable names are written in lowercase (a).
- A *composite object* is an object which is made up of one or more objects. Unit objects, lists, tagged objects, and programs are examples of composite objects.

In general, objects may be stored in variables or manipulated on the stack regardless of their type. Some HP 48 functions and commands perform different operations based on the type of object supplied as a parameter. For instance, the + function executes differently for strings (concatenates) than for real numbers (adds).

Object Evaluation

Evaluation of an object may be either implicit or explicit. Objects being entered on the command line, such as a real number or the name of a command such as +, are implicitly evaluated unless surrounding delimiters delay evaluation. An object on the stack may be explicitly evaluated by executing EVAL .

Evaluation results vary with the type of object:

- When a global variable name is evaluated, the contents of the variable are evaluated. To place a global variable name on the stack, enclose it in tick marks ('X').
- When a local variable name is evaluated, the contents of the local variable are recalled to the stack, but *not* evaluated. If a local variable contains a real number, the behavior is essentially the same as for a global variable, but if the local variable contains a program, the program will only be recalled to the stack. You can use a subsequent EVAL to evaluate the program.
- When a program is evaluated, global names are evaluated unless surrounded by ticks ('), the contents of local names are recalled to the stack, commands are executed, and all other objects are put on the stack.
- When an algebraic object is evaluated, the value it represents is computed and returned to the stack. Algebraic objects being evaluated obey rules of precedence – see the table on the next page.
- When a list is evaluated, global names are evaluated, programs are evaluated, commands are executed, and all other objects are put on the stack.
- All other objects are put on the stack.

Operator Precedence

Operator precedence controls the order in which calculations take place within an algebraic expression. Functions with the highest precedence (1) are evaluated before those with the lowest precedence (11). The evaluation order is left–to–right for operators having the same precedence. For instance, in the expression `'3+5*7'`, the multiply operation takes precedence over the add, resulting in the answer 38, whereas the answer would be 56 if evaluated from left to right.

Level	Operation
1	Expressions within parentheses
2	Functions
3	! (factorial)
4	Power (^) and square root ($\sqrt{\ }$)
5	Negate (–), multiply (*), divide (/)
6	Add (+) and subtract (–)
7	Relational operators (= =, ≠, <, >, ≤, ≥)
8	AND and NOT
9	OR and XOR
10	Left argument for | (where)
11	=

Object Types

Different object types may be distinguished in the stack display through their *delimiters* – characters that are unique to that type of object. For instance, strings are surrounded by quote marks ("), and programs are contained in French quotes (« »).

HP 48 objects are identified as follows:

Type	Object	Example
0	Real number	`1.2345`
1	Complex number	`(2.3,4.5)`
2	String	`"ABC"`
3	Real array	`[ 1 2 3 ]`
4	Complex array	`[ (1,2) (3,4) ]`
5	List	`{ "ABC" Var }`
6	Global name	`X`
7	Local name	`y`
8	Program	`« A 2 + »`
9	Algebraic	`'X=Y^2'`
10	Binary integer	`# 247d`
11	Graphics object	`Graphic 131 × 64`
12	Tagged object	`Dist: 34.45`
13	Unit object	`32_ft/s^2`
14	XLIB name	`XLIB 766 1`
15	Directory	`DIR ... END`
16	Library	`Library 766: ...`
17	Backup object	`Backup HOMEDIR`
18	Built-in function	`SIN`
19	Built-in command	`SWAP`
26	Library Data	`Library Data`

Related Commands: TYPE returns the *type* of object in level 1. VTYPE takes a variable name and returns the *type* of object in the variable, or −1 if the variable doesn't exist. TVARS takes a type number and returns a list of variables of that type in the current directory.

Real and Complex Numbers

Real Numbers. Real numbers have a 12-digit mantissa between 1 and 9.99999999999 and a 3-digit exponent between −499 and +499. During math operations, real numbers are expanded to have a 15-digit mantissa and a 5-digit exponent during the calculation, then rounded back to the 12-digit value when returned as results.

Complex Numbers. Complex numbers are represented by pairs of real numbers in parentheses: (2,3) (1.2,5). The Rectangular (X,Y) and Polar (r,θ) display modes (flags −15 and −16) control the appearance of a complex number on the stack, but do not affect the internal form. For instance, (2,3) is displayed in polar form as (3.60555127546,∡56.309932474).

Vectors and Matrices. Vectors and matrices may be composed of either real or complex numbers. Some examples:

`[ 1 2 ]`	*Real vector*

`[[ 1 2 ]`
` [ 3 4 ]]`
	Real matrix

`[[ (1,1) (1,2)`
` [ (2,1) (2,2) ]]`
	Complex matrix

Related Commands: The commands R→C and C→R convert between real and complex numbers or real and complex arrays. C→R, V→, and OBJ→ decompose a complex number to its real and imaginary parts. C→R separates a complex array into an array of real components and an array of imaginary components. OBJ→ separates a complex array into a series of complex numbers followed by a list containing the dimensions of the original array. If Complex Mode (flag −19) is set, →V2 creates a complex number.

RE returns the real component of a number or array; IM returns the imaginary component. ARG returns the polar angle θ of a coordinate pair (x,y). SIGN returns a unit vector in direction of the input argument (x,y).

Binary Integers

Binary integers are entered and displayed with a leading # delimiter and a trailing `b`, `d`, `h`, or `o` to indicate the base.

Examples: `#101101b` `#247d` `#7DACh`

The commands STWS and RCWS may be used to store or recall the wordsize, which may be up to 64 bits. The wordsize controls the interpretation of arguments and the results of arithmetic operations. For instance, if a binary integer is added to a real number, the real number is truncated to the current wordsize, and the result is a binary integer truncated to the current wordsize.

TRUTH TABLE

arg_1	arg_2	arg_1 AND arg_2	arg_1 OR arg_2	arg_1 XOR arg_2	NOT arg_1
1	1	1	1	0	0
1	0	0	1	1	0
0	1	0	1	1	1
0	0	0	0	0	1

Related Commands: The following commands are useful for working with binary integers: AND, B→R, NOT, OR, RCWS, RL, RLB, RR, RRB, R→B, SL, SLB, SR, SRB, STWS, and XOR.

Unit Objects

Unit objects are entered and displayed in the form: *number_units* where *number* is a real number and *units* is an algebraic expression containing unit names, prefixes, exponents and the operators `*`, `/`, and `^`. (A unit object may only contain one `/` operator.) During conversions, unit powers are rounded to integers MOD 256.

Examples:

```
32_ft/s^2
Density: 25_g/cm^3
```

Units in Menus. Unit objects in built-in menus or custom menus provide three types of functionality:

- Primary keys append the unit on the key to the numerator of the level 1 object.
- Left-shifted keys convert to the level 1 object to the unit on the key.
- Right-shifted keys append the unit on the key to the denominator of the level 1 object.

User-Defined Units. A user-defined unit may be created from any combination of the built-in units or other user-defined units. To create a user-defined unit, store the definition in a variable whose name is the name of the new unit.

For example, create the user-defined unit *week* by storing 7_d in the variable week. Executing UBASE on 2_week yields 1209600_s. The object 1_week stored in a custom menu will now behave like any other unit-related menu key.

Photometric Units. The numerical values of lumen (lm), lux (lx), phot (ph), and footcandle (fc) include a factor of $1/4\pi$ (steradian). To convert between these units and candela (cd), footlambert (flam), lambert (lam), or stilb (sb), do one of the following:

- Divide the expression including steradians by sr, the dimensionless unit for steradians, or
- Multiply the expression not including steradians by sr

Related Commands: The following commands are useful for working with unit objects: CONVERT, OBJ→, UBASE, UFACT, →UNIT, and UVAL.

Backup Objects

Backup objects are used to store backed-up data in independent memory (ports 1 or 2) or in port 0. A backup object may contain any object, including directory structures.

Backup Identifiers. The *contents* of a backup object are referenced by a *backup identifier* (eg: `:1:FRED`), which is a port-tagged name.

The wildcard `&` may be used for the port number for the commands RCL, EVAL, and PURGE. When the wildcard is evaluated, memory is searched in the order of ports 2, 1, 0, and then main memory for the first occurrence of the specified name.

If a backup object contains a directory structure, an object within that directory structure may be recalled or evaluated by specifying the path and name of the object in a port-tagged list. For instance, `:1:{ EEDIR FRED }` refers to the object FRED in a directory stored in backup object EEDIR in port 1.

Creating Backup Objects. A backup object is created by executing the STO command with the object in level 2, and the port-tagged name in level 1. For instance, the sequence `'FRED' RCL :1:BFRED STO` recalls the contents of variable FRED to the stack and creates a backup object called BFRED in port 1.

Recalling Backup Objects. The contents of a backup object may be recalled in two ways:

- Press [◀] [LIBRARY], PORT0, PORT1, or PORT2, then [▶] and the menu key for the backup object.
- Place the backup identifier on the stack and execute RCL.

Evaluating Backup Objects. The contents of a backup object may be evaluated in two ways:

- Press [◀] [LIBRARY], PORT0, PORT1, or PORT2 for the port number, then the menu key for the backup object.
- Place the backup identifier on the stack and execute EVAL. EVAL also accepts a list of backup identifiers.

Purging Backup Objects. To purge a backup object, place the backup identifier on the stack and execute PURGE. A backup identifier may be included in a list supplied to PURGE.

Related Commands: PVARS takes a port number as its argument and returns two results:

- Level 2 contains a list of backup objects and library IDs.
- Level 1 contains the type of memory in the port – `"SYSRAM"`, `"ROM"`, or a number showing the amount of available independent RAM.

Library Objects

Library objects are collections of one or more objects that generally extend the built-in command set. Libraries are referenced by a *library#* or a library identifier (**:** *port#* **:** *library#*), depending on the command. The title of the library may be displayed by pressing [↰] [REVIEW] in the LIBRARY menu.

Installing a Library. Library objects only extend the command set when they are stored in a port (0, 1, or 2) and *attached* to a directory in user memory. To use a library, perform the following:

- Store the library object in a port, such as port 0. For instance, if the library object is in level one of the stack, execute 0 STO.
- Turn the calculator off, then on again. The calculator will perform a system halt, which updates the system configuration to recognize the new library.
- Attach the library to the desired directory.
  - □ To attach a library to the current directory, enter the *library#* and execute ATTACH.
  - □ To detach a library from the current directory, enter the *library#* and execute DETACH.

Note: some libraries will automatically attach to the HOME directory. Any number of libraries may be attached to HOME, but only one library may be attached to each subdirectory.

Removing a Library. To purge a library, perform the following steps:

- Ensure that the library object does not appear on the stack as `Library nnn: ...` Either store the library in a variable or execute NEWOB to create a unique copy.
- If the library is attached to the HOME directory, enter the library#, such as `:2:272` and execute DETACH.
- Enter the library ID, such as `:2:272` and execute PURGE.

Variable Names

Variable names may contain letters, digits, and most characters. Names may not start with a digit, match a command name, or contain object delimiters or the characters + − * / ^ √ = < > ≤ ≥ ≠ ∂ ∫ ! space, comma, or @.

Reserved Variables. The HP 48 stores information for various commands in *reserved variables*. Reserved variables may reside in any directory, and may be used in more than one directory at a time.

Name	Description
ALRMDAT	Current alarm editing data
CST	Custom menu contents
EQ	Current equation for SOLVE and PLOT
IERR	Uncertainty of integration
IOPAR	I/O parameters
PICT	References the graphics display
PPAR	PLOT parameters
PRTPAR	PRINT parameters
der...	User-defined derivatives begin with *der*
n1, n2, ...	Integers created by ISOL
s1, s2, ...	Signs created by ISOL and QUAD
ΣDAT	Current statistical matrix
ΣPAR	Statistics parameters

Notes:

- The [↰] [I/O] SETUP menu *only* modifies the copy of *IOPAR* in the HOME directory.
- The print commands *only* modify the copy of *PRTPAR* in the HOME directory.
- *PICT* is not directory-dependent. It only refers to graphics display memory.

Symbolic Constants

The HP 48 has five constants which may be used in symbolic form or as approximate numerical values.

Name	Machine Value
π	3.14159265359
e	2.71828182846
i	(0,1)
MAXR	9.99999999999E499
MINR	1.E-499

System flags −2 and −3 control evaluation of symbolic constants:

Flag	Description	Clear	Set	Default
Symbolic Math Flags				
−2	Symbolic Constants	Symbolic form	Numeric form	Clear
−3	Numeric Results	Symbolic results	Numeric results	Clear

Memory Organization

Memory in the HP 48 is accessed in four-bit quantities (nibbles, or 1/2 bytes) within a 20-bit address space, yielding a 512K byte address space. The BYTES command, which returns the size and a checksum for an object, will sometimes show a size such as 106.5, reflecting that the object occupies 213 nibbles of memory.

System Memory

Memory in the HP 48 is organized as follows:

System ROM	The operating system resides in 256K bytes of read only memory (ROM). This command set may be extended through the use of library objects which reside in ROM or RAM (see *Library Objects*).
System RAM	There are 32K bytes of random access memory (RAM). Slightly less than 32K is available as user memory, as the rest is devoted to display memory and reserved system scratch and pointer memory.
Plug-in ROM	Plug-in ROM application cards, such as the HP 82211A HP Solve Equation Library, may extend the built-in command set.
Plug-in RAM	HP 48SX RAM may be extended by adding plug-in RAM cards that contain either 32K (HP 82214A) or 128K (HP 82215A). Plug-in RAM may be configured two ways (see below).

Configuring RAM Cards

Initial Configurations. Before a plug-in RAM card is used, some consideration should be given to its intended use. RAM cards may be configured two ways:

- *Independent* RAM may be thought of as an "electronic disk", which may be removed from the calculator. Individual objects or entire directories may be placed in independent RAM (see *Backup Objects* for more details). This configuration is most suitable for backing up data, "hiding" data from the HOME directory, or exchanging data with another calculator.
- *Merged* RAM extends the built-in RAM, creating more room for variables and directories, temporary objects, or graphics display area. To use a card in this manner, enter its port number and execute MERGE. Merged RAM may not be removed from the calculator unless the FREE command is used to free it. To free a card, make sure there is enough available memory to hold all your variables (including the contents of port 0), enter a blank list in level 2, the port number in level 1, and execute FREE.

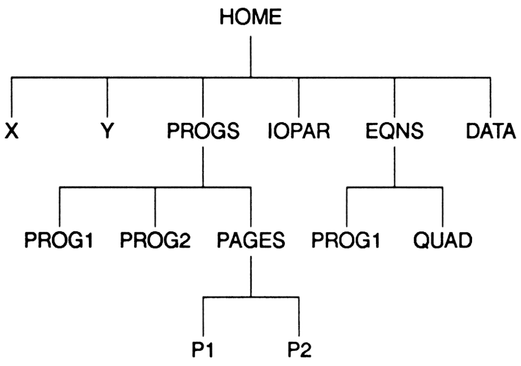
Changing Configurations. A merged RAM card may also be "converted" to an independent RAM card containing objects that were in port 0. To do this, enter a list containing the objects to transfer to independent RAM in level 2, the port number of the card in level 1, and execute FREE.

The reverse operation is also possible. An independent RAM card may be converted into merged RAM with the MERGE command. Any objects that were in the card will appear in port 0.

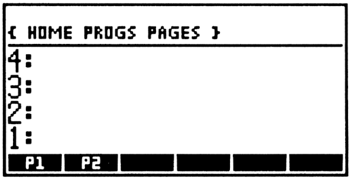
Understanding Port 0. Port 0 is a portion of built-in memory (which may include merged RAM cards) which behaves in the same manner as an independent RAM card (except that it is not removable). Port 0 may contain either library or backup objects. The amount of memory devoted to port 0 changes as objects are stored in it or purged from it.

User Memory

User memory may be organized into a tree structure of directory objects, which are implemented as variables stored in the HOME directory.



The status line displays the current directory path, and the VAR menu displays the current directory:



```
{ HOME PROGS PAGES }
4:
3:
2:
1:
```

In the example above, the current directory is PAGES, which contains variables P1 and P2.

Creating a Directory. A directory may be created with the command CRDIR. To store variables in the new directory move to the new directory by evaluating its name or pressing the corresponding key in the VAR menu.

Accessing Variables. When a variable name is evaluated, the current directory is searched first. If the variable is not found, its parent directories are searched in ascending order until the variable is found. In the example above, there are two variables named PROG1. Different directories may have variables of the same name.

Changing Directories. To change to a lower directory, simply evaluate its name. To return to the previous level, execute UPDIR (see *Menu Traversal Program*). Evaluating a list that starts with HOME followed by directory names can quickly change the current directory to any other place in user memory. For instance, if the current directory is PAGES, evaluating { HOME EQNS } will change the current directory to EQNS. A port-tagged path may be used for RCL and EVAL, but you must move to the target directory for STO.

Changing a Directory Name. To change the name of a directory or move the directory to another location, perform the following steps:

- Recall the directory to the stack
- Purge the old directory
- Move to the new location
- Enter the new name and execute STO.

Purging a Directory. The PURGE and PGDIR commands may be used to purge a directory. The PURGE command only removes empty directories; PGDIR removes a directory and its contents.

Saving User Memory. The commands ARCHIVE and RESTORE may be used to save and recover all of user memory (see *Data Transfer*).

Temporary Memory

The data stack in the HP 48 is actually a stack of pointers which refer to objects elsewhere in memory. Temporary memory is the calculator's "scratchpad". All objects that are not stored in a port or in a user variable reside in temporary memory. Many commands require temporary memory to construct intermediate objects or new objects returned as results to the stack.

Use of Temporary Memory. To understand temporary memory a little more, consider what happens when two math operations are perfomed. Enter the numbers 1.5 and 2.6 on the stack. These numbers now reside in temporary memory, referred to by pointers on the data stack. When the numbers are added, the result, 4.1, is a number in temporary memory referenced by a pointer in level 1 of the data stack. The objects 1.5 and 2.6 remain in temporary memory, referenced by pointers that save the Last Arguments.

Now add 2.8 to the result in level 1. The level 1 pointer on the data stack refers to the object 6.9 in temporary memory. The last arguments pointers now refer to the objects 2.8 and 4.1, and the objects 1.5 and 2.6 are no longer referenced.

Garbage Collection. From time to time the HP 48 will "hesitate" during an operation. This hesitation is usually caused by the removal of objects in temporary memory which are no longer being used. Objects which are no longer referenced continue to accumulate in temporary memory until memory has been filled. When memory is full, the calculator scans the objects in temporary memory, deleting those without references to them. This process, known as "garbage collection", is similar in concept to garbage collection in LISP.

A large number of pointers on the stack that point to temporary memory can slow down the garbage collection process to an uncomfortable degree. This occurs when there are a large number of objects on the stack, or an object has been extracted from a large list. List operations can be optimized by storing the lists in global variables, effectively moving the operations from temporary memory to user memory.

The MEM command returns the amount of available memory, forcing an initial garbage collection to return an accurate result. It may be helpful to insert the sequence MEM DROP to force garbage collection prior to speed-sensitive program sequences.

The NEWOB Command. The command NEWOB may be used to create a new copy of an object in temporary memory, whose only reference is on the data stack. In general, the system will perform an automatic NEWOB where it make sense. For instance, if you recall the contents of a variable to the stack and press [EDIT], the object will be copied to temporary memory before editing begins.

There are two uses for NEWOB:

- NEWOB "frees" an object that was extracted from a list. Consider the following program:

  ```
  « { "AB" "CD" "EF" } 2 GET »
  ```

  Level 1 of the data stack contains a pointer into the list, which still resides in temporary memory. Executing NEWOB now would create the unique object "AB" in temporary memory, and release the list for garbage collection. Note: set the Last Arguments flag (−55) to prevent the list from being references as a last argument.

- Recalling an object to the stack simply returns a pointer to the data stack. To purge a backup object from a port while retaining a copy in temporary memory, recall the object and execute NEWOB. Then the original object may be purged because there are no references to it.

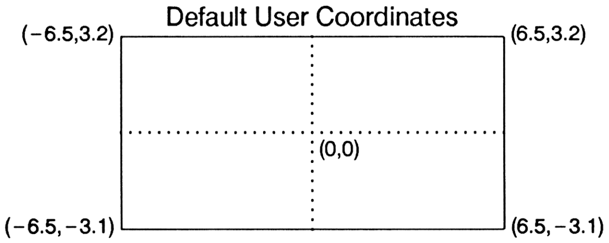
Graphics

The HP 48 display is a 131×64 pixel LCD which may present the stack or *PICT*, a portion of memory set aside for graphic displays.

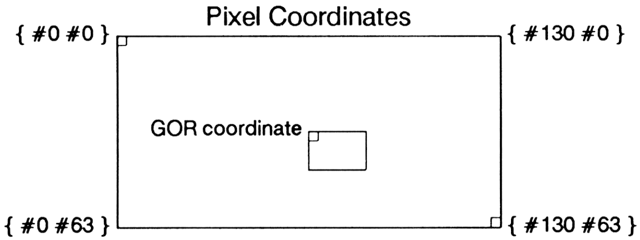
Graphics Coordinates

Two systems of coordinates may be used to manipulate *PICT* and graphic objects:

- User units, represented as complex numbers, are typically used to define the boundaries of plots. The first two entries in *PPAR* store the coordinates of the lower-left corner and upper-right corner of *PICT*. The default plot boundaries are (-6.5,-3.1) and (6.5,3.2). User-unit scaling information is stored in the reserved variable *PPAR*.



- Pixel coordinates are represented by a list containing two binary integers, { *#col* *#row* } . Graphics objects on the stack may only be described with pixel coordinates. The upper-left pixel is represented by { #0 #0 }.



Graphics objects added using GOR, GXOR, or REPL are located by their upper-left corner using either user or pixel coordinates. Note: the sequence *PICT* { #0 #0 } *grob* REPL is faster for animation than *grob PICT* STO.

Related Commands: The commands C→PX and PX→C convert between user-unit and pixel coordinates based on the dimensions in *PPAR*. The PDIM command changes the size of *PICT*.

C→PX			Command
User-unit to pixel coordinate conversion			
(x,y)	→	{ #col #row }	
PDIM			Command
Changes the size of *PICT*.			
(x_{min},y_{min}) (x_{max},y_{max})	→	*Changes PICT* relative to the *current user coordinates*	
#horizontal #vertical	→	*Does not affect current user coordinates*	
PMAX			Command
Sets the upper-right plot coordinates			
(x,y)	→		
PMIN			Command
Sets the lower-left plot coordinates			
(x,y)	→		
PX→C			Command
Pixel to user-unit coordinate conversion			
{ #col #row }	→	(x,y)	
SCALE			Command
Specifies x and y scale in units per 10 pixels			
x y	→		

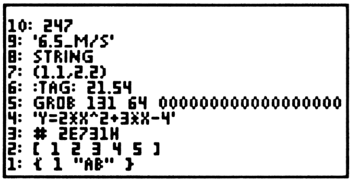
Other commands that affect scaling are AUTO, AXES, DEPND, INDEP, *H, and *W.

Stack View Program

The following stack-view program STKV displays up to ten levels of the stack simultaneously. The display mode, plot parameters, stack values and graphics picture are preserved. The system remains halted until ATTN is pressed, after which the program resumes to restore the original *PPAR* and *PICT*.

STKV   371.5 Bytes   Checksum #A1B7h

```
« IF DEPTH THEN                      Make sure stack is not empty
  PICT RCL PPAR → pict ppar          Preserve original PICT and PPAR
  « PICT PURGE                       Purge original PICT
    1 32 XRNG 1 64 YRNG              Set new X and Y ranges for stack
    1 DEPTH 1 - 10 MIN DUP           Determine current stack height
    IF 8 >                           If greater than 8, text row height
    THEN 6 1                         is 6 and text size is 1
    ELSE 8 2                         Otherwise, text row height is 8
    END → rowht tsize                and text size is 2
    « FOR i PICT 1 i rowht           Loop for the no. of stack levels:
         * R→C RCLF STD i            Use STD display mode to
         ": " + SWAP STOF            build stack level identifier
         i 3 + PICK →STR +           Add stack value to identifier,
         tsize →GROB GOR             and add to picture
      NEXT                           End loop
      { } PVIEW                      Display PICT, wait for [ATTN]
      'PPAR' PURGE ppar              Purge new PPAR
      IF 'PPAR' SAME NOT             Did PPAR exist before?
      THEN ppar 'PPAR' STO           Yes, store old value
      END
      pict PICT STO                  Restore original PICT
    »
  »
  END
»
```



```
10: 247
9: '6.5_M/S'
8: STRING
7: (1.1,2.2)
6: :TAG: 21.54
5: GROB 131 64 000000000000000000
4: 'Y=2*X^2+3*X-4'
3: # 2E731H
2: [ 1 2 3 4 5 ]
1: { 1 "AB" }
```

GROB Structure

A graphics object is structured as follows:

<header> <length> <height> <width> <data...>

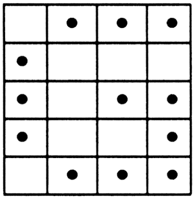
header	This is a five-nibble* field that distinguishes a graphics object from any other object type, and has a fixed value of #02B1Eh.
length	This field is a five-nibble quantity that contains the distance in nibbles from start of length field to the nibble past the end of the object. This length is #Fh + the number of data nibbles.
height	This field is a five-nibble quantity that specifies the height of the graphics image in pixels.
width	This field is a five-nibble quantity that specifies the width of the graphics image in pixels.
data	The data nibbles begin at the upper-left corner of the graphics object and proceed left-to-right, top-to-bottom. Each row must contain an integral number of bytes, so the data may be padded with garbage bits. The bits in each nibble are written in reverse order, so the leftmost displayed pixel in a nibble is represented by the least-significant bit of the nibble.

If you are preparing a graphics object on a personal computer, remember that the HP 48 CPU reads data from memory into registers in reverse order, so the first four fields are written backwards. For example, the header is written E1B20.

* A nibble is 1/2 byte.

Graphics objects may be entered into the command line on the HP 48. To enter a blank graphics object, type GROB *width height*, where *width* and *height* specify the size in pixels.

Examples: To enter a graphics object which represents "G" in the small font, type GROB 4 5 E010D090E0 .



On a personal computer, the graphics object looks like this:

```
E1B20   B1000   50000  40000  E010D090E0
header  length  width  height    data
```

In the second example consider a blank graphics object that is the size of the display with the "G" from above in the upper-left corner. The graphics object looks like this on a personal computer:

```
E1B20                                      header
F8800                                      length
04000                                      height
38000                                      width
E0000000000000000000000000000000000        row 1
10000000000000000000000000000000000        row 2
D0000000000000000000000000000000000        row 3
90000000000000000000000000000000000        row 4
E0000000000000000000000000000000000        row 5
00000000000000000000000000000000000        row 6
        ... 2176 total data nibbles         ...
00000000000000000000000000000000000        row 64
```

The fields for the example on the previous page are derived as follows:

- The display width is 131 columns = 83h pixels, or 17 bytes or 34 nibbles.
- The display height is 64 rows = 40h pixels.
- The data length is bytes–per–row x rows = 2176 nibbles. The length field is calculated as 2176 + 15 = 2191d = 88Fh.

PPAR

The reserved variable *PPAR* (which may exist in every directory) contains scaling information and plot specifications.

PPAR →

{ (x_{min}, y_{min}) (x_{max}, y_{max}) indep resolution (x_{axis}, y_{axis}) ptype depend }

Parameter	Description	Default
(x_{min}, y_{min})	Lower–left pixel coordinates	(−6.5, −3.1)
(x_{max}, y_{max})	Upper–right pixel coordinates	(6.5, 3.2)
indep	Independent var for horizontal axis	X
resolution	Real positive integer for user–unit point spacing, or binary integer for pixel spacing (0=every column). Specifies the bar width for BAR plots or the bin width for HISTOGRAM plots.	0
(x_{axis}, y_{axis})	Axes intersection coordinates	(0, 0)
ptype	Plot type: FUNCTION, CONIC, POLAR, BAR, PARAMETRIC, HISTOGRAM, SCATTER, TRUTH	FUNCTION
depend	Dependent variable	Y

Statistics Data

Data used by the STAT application resides in or is named by the reserved variable *ΣDAT*. Statistics data may be entered from the stack one point at a time using the Σ+ command, or an entire matrix can be stored in *ΣDAT* using the [↰] [STAT] NEW command. The command EDITΣ may be used to edit *ΣDAT* using the MatrixWriter.

X Σ+	→	Append one data point with one coordinate value
Σ-	→	Reverses the effect of the last Σ+
$[X_1\ X_2 \ldots X_m]$ Σ+	→	Append one data point with *m* coordinate values
$[[X_{21} \ldots X_{2m}]$ … $[X_{n1} \ldots X_{nm}]]$ Σ+	→	Append *n* data points with *m* coordinate values

ΣDAT Statistics Matrix						
Data Point	**Coordinate Number**					
	1	**2**	**3**	**4**	**...**	**m**
1	X_{11}	X_{12}	X_{13}	X_{14}	...	X_{1m}
2	X_{21}	X_{22}	X_{23}	X_{24}	...	X_{2m}
3	X_{31}	X_{32}	X_{33}	X_{34}	...	X_{3m}
...	...	...	...	...	...	...
n	X_{n1}	X_{n2}	X_{n3}	X_{n4}	...	X_{nm}

ΣPAR

The reserved variable *ΣPAR* contains plot and scaling information. Each directory may contain a unique *ΣPAR*. The entries for the independent and dependent columns may be set using the COLΣ command.

ΣPAR →

{ indep dep intercept slope model }

Parameter	**Description**	**Default**
indep	Independent column number	1
dep	Dependent column number	2
intercept	Intercept of current regression model	0
slope	Slope of current regression model	0
model	Current model: LINFIT, EXPFIT, PWRFIT, or LOGFIT	LINFIT

Data Transfer

Any named object, such as a variable, backup object, or complete directory, may be transferred to another HP 48 or a computer. A complete backup of user memory may also be transferred to another HP 48 or a computer.

Pathways

There are three methods of transferring data between the HP 48 and another HP 48 or computer:

- Objects may be transferred between HP 48s using the infrared (IR) link. The IR link is fixed at 2400 baud, no parity, and may be used to transfer data in either ASCII or binary mode.
- Objects may be transferred between a computer and an HP 48 using the serial (wire) link. The wire link may be configured to support a variety of baud rates and parity options. The Kermit protocol provides the most reliable transfer mechanism.
- Plug-in RAM cards may be configured as independent memory and exchanged between HP 48s. The commands FREE and MERGE are used to configure RAM cards. Only library and backup objects can reside in independent memory.

Kermit Protocol

The *Kermit* file transfer protocol ensures correct data transmission between two HP 48 calculators or an HP 48 and a computer. Kermit was developed at the Columbia University Center for Computing Activities. Detailed information about Kermit is available in a book by Frank da Cruz, *KERMIT, A File Transfer Protocol*, 1987, Bedford, MA (Digital Press). For 9600 baud transfers, it's best to disable the updating clock display.

Kermit Configurations. Kermit protocol provides two basic configurations for data transfer:

Local/Local	Commands must be entered on both machines to effect a transfer: a SEND command must be issued on the sender, and a RECEIVE (RECV or RECN on the HP 48) command must be issued on the receiver. New commands must be issued for each object transferred. (Some implementations of Kermit permit "wildcard" characters to send a series of files with one command.)
Local/Server	One machine is placed in *server* mode, which acts upon commands received from the sender. The server:

- Transmits an object when it receives a GET command with a file name.
- Receives an object when it receives a SEND command.
- Exits Kermit when it receives a FINISH command.

The server may respond to multiple transfer requests without keyboard intervention.

Remote Kermit Operation. The HP 48 can respond to several Kermit commands when in server mode. These commands initiate actions, list variables, or transfer data.

GET: The Kermit command GET *name* instructs the HP 48 server to transmit the contents of the named variable to the computer.

SEND: The Kermit command SEND *name* instructs the HP 48 server to receive the contents of the named computer file and store them in a variable of the same name.

REMOTE DIR: The Kermit command REMOTE DIR (packet GD) causes the HP 48 server to reply with a separate line for each variable in the current directory. Each line contains the variable name, length in bytes, type, and a decimal checksum. Examples:

Name	Length	Type	Checksum
X	16	Real Number	7537
EQ	40	Algebraic	14632
CLK	6876	Directory	28291
IOPAR	29.5	List	7079

REMOTE HOST: The Kermit command REMOTE HOST (C "*host-command*" packet) may be used to execute HP 48 commands from the computer. After the command has been executed, the HP 48 replies by returning the stack contents. The stack is formatted in a manner similar to the PRSTC (print stack compact) command. For instance, to add two numbers on the HP 48, type "REMOTE HOST 2 3 +". Assuming that the stack was empty before, the HP 48 replies with the string `"1:        5"`. If the stack is empty, the HP 48 replies `Empty Stack.`

FINISH: The Kermit command FINISH transmits the GF packet to the HP 48 to turn off server mode on the HP 48. The GL packet, associated with logout commands, has the same effect.

HP 48 ⟷ HP 48

To transfer an object between two HP 48s, perform the following:

- Use the [⬅] [I/O] SETUP menu to set IR transmission mode and type 3 checksums.
- Set the sender to the directory containing the variables to send.
- Set the receiver to the directory that will receive the variables.

Local/Local Configuration

1. On the receiver, execute RECV to store the incoming variable under the sender's name, or enter a name and execute RECN to rename the incoming variable.
2. On the sender, enter the variable name and execute SEND.
3. Repeat 1 and 2 for each additional variable.

Local/Server Configuration

1. On the *server* HP 48, execute SERVER ([➡] [I/O]).
2. On the *local* HP 48:
   - To send variables to the server, enter the variable name and execute SEND.
   - To receive variables from the server, enter the variable name and execute KGET.
3. After all variables have been transferred, execute FINISH on the local HP 48 or press [ATTN] on the server.

HP 48 ⟷ Computer

To transfer objects between the HP 48 and a computer, perform the following:

- Use the [↰] [I/O] SETUP menu to set wire transmission mode, the baud rate, parity, and checksum settings.
- Set the HP 48 to the directory which will send or receive objects.

Local/Local Configuration

1. Issue the receive command:

HP 48:	Execute RECV or enter the variable name and execute RECN.
or **Computer:**	Issue the RECEIVE command.

2. Issue the send command:

HP 48:	Enter the variable name and execute SEND.
or **Computer:**	Issue the SEND *file-specifier* command.

3. Repeat 1 and 2 for each additional file, then execute CLOSEIO on the HP 48 to save battery power.

Local/Server Configuration

1. Set the server operation:

HP 48:	Execute SERVER ([↱] [I/O]).
or **Computer:**	Execute the Kermit Server command.

2. On the local device:
   - To send a variable, enter the variable's name and execute the SEND command.
   - To receive the contents of a variable on the server, enter the variable name and execute GET or KGET.
3. After all variables have been transferred, execute FINISH on the local device and CLOSEIO on the HP 48 to save battery power.

Backing Up the HP 48

The ARCHIVE and RESTORE commands may be used to save and recover the entire contents of user memory on a computer.

Note: The system and user flag settings may be preserved by executing RCLF and storing the flags in a variable. After doing a restore, recall the contents of the variable and execute STOF.

To back up all of user memory to a computer, perform the following steps:

- Connect the HP 48 and the computer.
- Use the ⬅ I/O `SETUP` menu to set wire transmission mode, the baud rate, parity, and checksum settings.
- *Optional*: Execute RCLF and store the flags in a variable.
- Enter the object `:IO:`*name*, where *name* is the computer file name that will contain the HP 48 image. For 9600 baud transfers, it's best to disable the updating clock display.
- Issue the Kermit RECEIVE command on the computer.
- Execute ARCHIVE on the HP 48.

Restoring the HP 48

Caution: The RESTORE command erases the *entire* contents of user memory!

To restore the user memory image from a computer, perform the following steps:

- Be sure there is enough user memory available to hold the incoming file. Since the RESTORE will replace all of user memory, you might as well execute CLVAR.
- Connect the HP 48 and the computer.
- Transfer the file containing the memory image to the HP 48 the same way as for any file.
- Put the file name on the stack and execute RCL. This puts `Backup HOMEDIR` in level 1.
- Execute RESTORE.
- *Optional*: Recall your variable containing the user and system flags and execute STOF.

ASCII File Transfer

An ASCII file generated on a computer provides an alternative method for entering data or a large program in the HP 48. To ensure that the data is interpreted correctly by the receiving HP 48, the following header string should be included which indicates the expected modes:

```
%%HP: T(translation)A(angle-mode)F(fraction-mark);
```

The codes are defined as follows:

Code	Purpose	Settings	Default
T	See *Character Translations*	0, 1, 2, or 3	1
A	Sets the angle mode	D, R, or G	D
F	Sets the fraction mark	, *or* .	.

The HP 48 will ignore text after the `@` character at the end of a line in the computer file.

Example: The following text on a computer may be transferred to the HP 48 in ASCII mode to create a program that returns the area and volume of a sphere given its radius. Notice the use of character translations to represent various HP 48 characters:

```
%%HP: T(3)A(D)F(.);
\<< \-> r \<<  @ Comment information
  4 \pi \->NUM * r 2 ^ * "Area" \->TAG
  4 3 / \pi \->NUM * r 3 ^ * "Volume" \->TAG
  \>>
\>>
```

On the HP 48, the program looks like this:

```
« → r
  « 4 π →NUM * r 2 ^ * "Area" →TAG
    4 3 / π →NUM * r 3 ^ * "Volume" →TAG
  »
»
```

Character Translations

When data is transferred between the HP 48 and a computer using translate codes 2 (000→159) or 3 (000→255), conversions are used to represent some characters.

For data being transferred to a computer with translate codes 2 or 3, each \ is replaced with \\. For data being transferred to the HP 48, characters may be converted using a text conversion or *xxx*, where *xxx* is the three-digit (decimal) character code.

The following table shows the text conversions for characters above code 127.

NUM	HP 48	ASCII	NUM	HP 48	ASCII
128	∡	\<)	148	η	\Gn
129	x̄	\x-	149	θ	\Gh
130	∇	\.V	150	λ	\Gl
131	√	\v/	151	ρ	\Gr
132	∫	\.S	152	σ	\Gs
133	Σ	\GS	153	τ	\Gt
134	▶	\\\|>	154	ω	\Gw
135	π	\pi	155	Δ	\GD
136	∂	\.d	156	Π	\PI
137	≤	\<=	157	Ω	\GW
138	≥	\>=	158	■	\■
139	≠	\=/	159	∞	\oo
140	α	\Ga	171	«	\<<
141	→	\->	176	°	\^o
142	←	\<-	181	µ	\Gm
143	↓	\\\|v	187	»	\>>
144	↑	\\\|^	215	×	\.x
145	γ	\Gg	216	Ø	\O/
146	δ	\Gd	223	ß	\Gb
147	ε	\Ge	247	÷	\:-

IOPAR

The reserved variable *IOPAR* may only reside in the HOME directory. Other variables of the same name in subdirectories will be ignored by the I/O commands.

IOPAR →

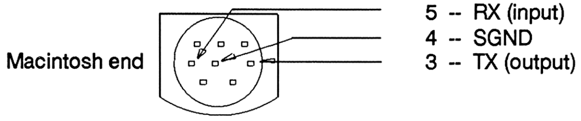
{ baud parity receive-pacing transmit-pacing checksum translate-code }

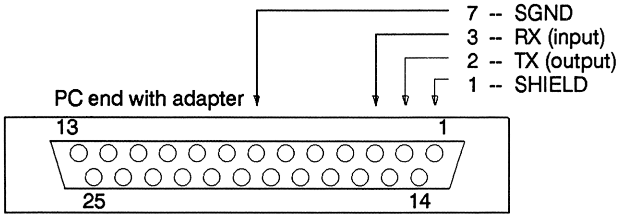
Parameter	**Description**	**Default**
baud	1200, 2400*, 4800, or 9600	9600
parity	0=none*, 1=odd, 2=even, 3=mark, 4=space	None
	Negative parity value = transmit only	
receive-pacing†	Value ≠ 0 sends XOFF if HP 48 buffer full	0
transmit-pacing†	Value ≠ 0 stops transmission if XOFF received	0
checksum	1=1 digit arithmetic, 2=2 digit arithmetic, 3=CRC	3
translate-code	0=none, 1=LF to CR-LF, 2=128-159, 3=128-255	1

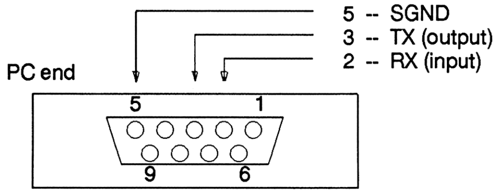
* IR is 2400 baud, no parity only   † Not used by Kermit

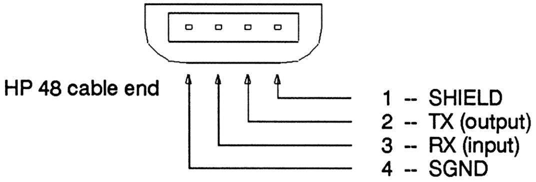
Cables

The Serial Interface Kits include a serial cable for an IBM-compatible personal computer (HP 82208A) or an Apple Macintosh computer (HP 82209A), and a copy of Kermit that can run on the host computer.









Menus

Custom Menus

A custom menu may be created using a list of objects supplied to the MENU or TMENU commands.

{ Key_1 Key_2 Key_3 ... }

The objects that define each key in the menu may range in complexity from a real number to a list definition with a graphics object for the menu key label and separate actions for the primary and left– or right–shifted planes.

The Variable CST. The MENU command stores the definition in the reserved variable *CST* and immediately displays the menu. Each directory may have a different variable *CST*. A name may be stored in *CST* which references a variable containing the menu definition. The TMENU command does not affect *CST*.

Menu Contents. Menus may contain any object, but the functionality of the key is determined by the type of the object:

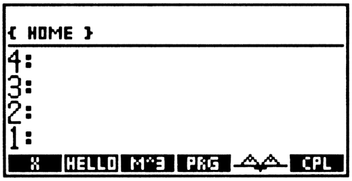
- Names work the same way as the VAR menu.
- Keys with string definitions echo the string.
- Directory names change to the directory.
- Unit objects act as unit catalog entries:
  - □ Primary keys append the unit on the key to the numerator of the level 1 object.
  - □ Left–shifted keys convert the level 1 object to the unit on the key.
  - □ Right–shifted keys append the unit on the key to the denominator of the level 1 object.
- Backup objects act like the port 0, 1, and 2 menus.
- Labeled objects can be used to identify menu key actions and can provide optional shifted functionality.

Labels. A menu key can have a label that is different than its key action. The most versatile key definition provides separate objects for the label, primary, left-shifted, and right-shifted actions. Either a string or a graphics object 8 rows high by 21 columns wide may be supplied as the label.

Example: The following list contains a menu definition for six keys: a variable, string, unit object, labeled program, a definition that uses a graphics object for the menu label, and labeled key definition with shifted functionality:

MENUEX 226.5 Bytes Checksum #C051h

```
{
  X
  "HELLO"
  1_m^3
  { "PRG" « 2 * 3 + » }
  { GROB 21 8 0000000404000A0A0005151080A020FFFFF100F100004000
   "Kilroy was here!"
  }
  { "CPL" {
          « CPL »           primary action
          « 'CPL' STO »     left-shifted action
          « 'CPL' RCL »     right-shifted action
          }
  }
}
```


```
{ HOME }
4:
3:
2:
1:
```

Menu Traversal Program

The commands RCLMENU and UPDIR may be used to traverse the built-in menu trees as well as the directory tree in the VAR menu. This program allows automatic movement from any menu to its parent (if one exists) or to the last menu viewed if no parent exists (see *Menu Numbers*). If the parent menu key leading to the currently displayed menu is on a page beyond page 1 (such as in the UNITS submenus which have parents in pages 1 through 3 of the main UNITS menu), this routine will return to the correct originating page of the parent. Menu numbers greater than 59 have the LIBRARY menu as their parent.

The program is based on a 61-element list called PARENT. Each element *n* of the list has the value of the menu number and page of the parent corresponding to menu *n* for menus 1 through 59. The first element accounts for a zero result from RCLMENU. The last element accounts for LIBRARY submenus.

If UP is assigned to [↰] [UP], it replaces the normal action of that key when the HP 48 is in USER mode. To make this assignment, execute 'UP' 31.2 ASN .

PARENT (61-element list) 456 Bytes Checksum #8DB8h

```
{ 0 0 0 0 3 3 3 3 3 3 0 10 10 10 10 10 10 0
  0 18 0 0 0 0 0 24 24 24 0 0 29 0 31 31 0
  0 35 35 37 35 0 40.04 0 42 42 42 42 42 42
  42.02 42.02 42.02 42.02 42.02 42.02 42.03
  42.03 42.03 42.03 42 24 }
```

UP 89 Bytes Checksum #235Bh

```
« RCLMENU IP 1 + 61 MIN DUP
  IF 3 SAME
  THEN DROP UPDIR
  ELSE PARENT SWAP GET MENU
  END »
```

User Keys

Variables, programs, commands, or strings may be assigned to any key on the HP 48. When 1-User or User mode is active, these objects are evaluated in place of the standard key definitions.

The ASN and STOKEYS commands may be used to assign an object to a key. The command RCLKEYS recalls the current key assignments, and DELKEYS deletes one or more assignments. These commands are shown on the next page.

Setting User Mode

1-User mode may be set by pressing [↰] [USR]. 1-User mode remains in effect for only one operation. User mode may be locked by pressing [↰] [USR] twice or by setting flag -62. When flag -61 is set [↰] [USR] toggles user mode, and 1-User mode is not available.

Key Locations

The notation *rc.p* specifies the location of a key where *r* is the row, *c* is the column, and *p* is the plane.

p	**Primary Planes**	*p*	**Alpha Planes**
0 or 1	Unshifted	4	Alpha
2	Left-shifted	5	Alpha left-shifted
3	Right-shifted	6	Alpha right-shifted

Examples: the ENTER key is 51.0 (or 51), the PURGE key is 54.2, and the alpha right-shifted CST key is 23.6.

Standard Keys

When User mode is set, the standard key definitions apply to all keys which have not been reassigned. The standard key definitions may be disabled by using supplying the S parameter to the DELKEYS command. The symbol S refers to standard key definitions. An individual standard key definition may be reactivated by supplying SKEY as the assigned object for ASN. All standard keys may be reactivated by supplying SKEY to STOKEYS.

Related Commands:

ASN			Command
Make a single user-key assignment			
	object rc.p	→	
	'SKEY' rc.p	→	*Reactivates standard key*
DELKEYS			Command
Clears user-key assignments			
	rc.p	→	*Clears a single key*
	{ $rc.p_1$ $rc.p_2$ ... }	→	*Clears a list of keys*
	S	→	*Clears standard key definitions*
	{ S $rc.p_1$ $rc.p_2$ ... }	→	*Clears list of keys & std key defs*
	0	→	*Clears all user keys*
RCLKEYS			Command
Lists user-key assignments. S indicates standard keys are active.			
		→	{ obj_1 $rc.p_1$ ... obj_n $rc.p_n$ }
		→	{ S obj_1 $rc.p_1$ ... obj_n $rc.p_n$ }
STOKEYS			Command
Makes multiple user-key assignments. Including S activates standard key definitions.			
	S	→	
	{ obj_1 $rc.p_1$ ... obj_n $rc.p_n$ }	→	
	{ S obj_1 $rc.p_1$ ... obj_n $rc.p_n$ }	→	

Key Assignment Program

A simple program, `« 0 WAIT ASN »`, may be used to assign an object to a key. Store the program in a user variable (or assign it to a key!). Place the object to assign in level one, execute the program, and press the key to be assigned.

Programming

Program Structure

In the simplest form, a program is a collection of commands or functions enclosed by program delimiters (« »). A simple example returns the area of a circle given its radius in level 1:

```
« 2 ^ π →NUM * »
```

Programs which are more involved may use *local variables* to avoid potential conflicts with global variables. The formal syntax for programs using local variables is:

« → *local-names defining-procedure* »

Local variables exist in a local environment during execution of the defining procedure and take precedence over global variables of the same name when evaluated. Values for the local variables may be established at the start of the program, prior to the →. The defining procedure may be either an algebraic expression or a program.

Example: Suppose the stack contains 3 in level 3, 2 in level 2, and 1 in level 1. The following programs produce the same result (17) by first assigning the values to local variables *x, y,* and *z*:

```
« → x y z '(x*y+z)*2+x' »

« → x y z
  « x y * z + 2 * x + »
»
```

When a local variable is evaluated, it *only* recalls the contents of the variable. This is similar to evaluating global names that contain data objects. However, if the local variable contains a program, it can only be executed by an explicit EVAL.

User – Defined Functions

User-defined functions may be used to extend the function set of the HP 48. A user-defined function takes its arguments from the stack and must return exactly one result to the stack. The arguments may be either algebraic or numeric.

The syntax of a user-defined function must be exactly:

« → *local-names defining-procedure* »

User-defined functions created with the DEFINE command use an algebraic expression as the defining procedure. If the defining procedure is a program, the program must remove all arguments from the stack and return one real number.

The DEFINE command simplifies the creation of a user-defined function by converting an expression in the form '*name*(*arguments*)=*expression*' into a named program that consists of a local variable structure and an algebraic expression.

Example: Create a function POLY(x) = $2x^2 + 4x + 7$. Enter the expression `'POLY(x)=2*x^2+4*x+7'` and execute DEFINE. The variable POLY in the VAR menu now contains the program:

```
« → x '2*x^2+4*x+7' »
```

If the number 8 is in level 1, executing POLY yields 167. Assuming that the variable S is undefined, POLY('S+5') yields the expression `'2*(S+5)^2+4*(S+5)+7'`.

Example: Create a function PTHG(x,y) = $\sqrt{x^2 + y^2}$. Enter the expression `'PTHG(x,y)=√(x^2+y^2)'` and execute DEFINE. The variable PTHG in the VAR menu now contains the program:

```
« → x y '√(x^2+y^2)' »
```

Looping Structures

Program loops are useful for repetitive execution of a procedure. There are two general classes of loops:

- *Definite loops* execute a *loop-clause* at least once, and execute a predefined number of iterations.
- *Indefinite loops* execute a *loop-clause* repeatedly until a *test-clause* returns a true (non-zero) result. One form of an indefinite loop may not execute at all if an initial test fails.

Definite Loops. There are two types of definite loops, both of which can have an increment of either 1 or *n*:

start finish FOR *index loop-clause* NEXT

start finish FOR *index loop-clause increment* STEP

start finish START *loop-clause* NEXT

start finish START *loop-clause increment* STEP

In each case the *start* and *finish* values are taken from the stack and are no longer available to the program. The *index* is a local variable that may be referenced in the loop clause just like any other local variable. The *increment* is also taken from the stack. This syntax shows it being put there by the program, but it can be calculated also.

	Increment = 1	**Increment = *n***
Index	FOR ... NEXT	FOR ... *n* STEP
No Index	START...NEXT	START... *n* STEP

The differences are:

- FOR loops keep their index in a local variable which is available to the loop-clause. An early exit may be taken from a FOR loop by one of the following two methods:
  - □ Store MAXR in the index for loops with a positive step.
  - □ Store −MAXR in the index for loops with a negative step.
- START loops save memory and execute faster than FOR loops for applications where access to the index is not needed and the increment will always be 1.
- Loops ending with STEP may have a varying increment. When STEP is executed, the increment is added to the index. The loop will repeat under the following conditions:
  - □ The increment is positive and the index is less than the finish value.
  - □ The increment is negative and the index is greater than the finish value.
- Loops ending with NEXT execute faster than those ending with STEP, because the increment value is always 1.

Examples:

« 1 10 START *loop-clause* NEXT »
Executes *loop-clause* 10 times.

« 1 20 FOR x *loop-clause* NEXT »
Executes *loop-clause* 20 times; *x* is the index.

« 1 10 START *loop-clause* 2 STEP »
Executes *loop-clause* 5 times.

« 1 20 FOR x *loop-clause* 2 STEP »
Executes *loop-clause* 10 times; *x* is the index.

Indefinite Loops. There are two forms of indefinite loops:

- `DO` *loop-clause* `UNTIL` *test-clause* `END`

  DO loops execute at least once. The placement of UNTIL is unimportant since the test occurs at the end, but by convention is placed between the loop and test clauses to improve legibility.

- `WHILE` *test-clause* `REPEAT` *loop-clause* `END`

  WHILE loops never execute if the test-clause returns an initial false (zero) result. The placement of REPEAT is important, as it isolates the test clause, which usually executes one time more than the loop clause.

Loop Counters. The commands INCR and DECR may be used at any time to increment or decrement a real number stored in a variable.

The command INCR takes a local or global variable name, increments its contents, and returns the new value to the stack. For instance, if x contains 23, `'x' INCR` stores 24 in x and returns 24 to the stack. DECR behaves the same way as INCR, but decrements the variable's contents.

Examples: The first program (46 bytes, checksum #FD95h) *always* prints at least one carriage-right, up to the number of carriage-rights specified in level 1. The second program (48.5 bytes, checksum #FEDCh) prints the number of carriage-rights specified in level 1.

```
« → x
  « DO x DECR CR UNTIL x NOT END »
»
```

```
« → x
  « WHILE x REPEAT x DECR CR END »
»
```

Conditional Structures

IF Structures. The IF structures perform a test and execute a *true-clause* if the test is true or a *false-clause* if the structure includes ELSE.

`IF` *test-clause* `THEN` *true-clause* `END`	`IF` *test-clause* `THEN` *true-clause* `ELSE` *false-clause* `END`

```
IF
        test-clause
THEN
        true-clause
END
```

```
IF
        test-clause
THEN
        true-clause
ELSE
        false-clause
END
```

Example: This program (82.5 bytes, checksum #ACF0h) stores a value from the stack into variable a and returns .35*a or .45*a if a > 10.

```
« → a
  « IF 'a>10'
    THEN .45
    ELSE .35
    END
    a *
» »
```

IFT and IFTE. IFT and IFTE may be used as as commands, taking their arguments from the stack. IFTE may also be used in an algebraic expression.

`IFTE(`*test-clause*`,` *true-clause*`,` *false-clause* `)`

Level	IFT	IFTE
3:		*test-result*
2:	*test-result*	*true-clause*
1:	*true-clause*	*false-clause*

CASE Structures. The CASE...END structure combines a series of IF...THEN structures that ends when the first true condition has been met. A "default" clause may be placed before the END command which is executed if none of the conditions have been met.

```
CASE
    test-clause       THEN    true-clause    END
    test-clause       THEN    true-clause    END
    ...
    test-clause       THEN    true-clause    END
    default-clause
END
```

Example: This program (127 bytes, checksum #A7F1h) accepts an object and issues an error for non-real types, executes the procedure *Xneg* for numbers less than zero, *Xzero* for numbers equal to zero, or *Xpos* in the default case.

The type for a real number is zero, so a non-real object generates a true condition. In this case the command DOERR will issue message #202h, "Bad Argument Type".

```
« → x
  « CASE
      x TYPE THEN # 202h DOERR END
      'x<0'  THEN Xneg END
      'x==0' THEN Xzero END
      Xpos
    END
  »
»
```

Error Trapping

The IFERR structure is useful for trapping anticipated errors. The *trap-clause* is executed first, and if no error is encountered an optional `ELSE` *normal-clause* is executed. If an error occurs within the trap clause, the remainder of the trap clause is bypassed and the *error-clause* is executed. Note that the Last Arguments flag (flag −55) controls whether the arguments that generated the error will be returned to the stack.

`IFERR`
    trap-clause
`THEN`
    error-clause
`END`
	`IFERR`
	    trap-clause
	`THEN`
	    error-clause
	`ELSE`
	    normal-clause
	`END`

Example: This program (65 bytes, checksum #15A4h) takes the a port number *p* from the stack and returns the port variables. If port *p* is empty, the program returns "".

```
« → p
  « IFERR PVARS
    THEN IF -55 FC? THEN DROP END ""
    END
  »
»
```

Error Interpretation. The commands ERRM and ERRN return the most recent error message and error number. ERR0 clears the error number. These commands may be useful in an error clause for taking specific action for different kinds of errors.

User-Defined Errors. The command DOERR accepts either a system error number or a string. If the error number is zero, the action is equivalent to pressing [ATTN], and ERRM and ERRN are set to "" and 0. If a string is supplied, the string will be returned by ERRM and the error number will be set to `#70000h`.

Data Entry

A program may halt to obtain user input using a variety of techniques. These techniques have varying levels of restrictions on keyboard and stack operations:

- Execute HALT. The program resumes when the command CONT is executed or the user presses [CONT]. The stack is available in this state.
- Execute PROMPT. The program displays a message and halts until CONT is executed or the user presses [CONT]. This is equivalent to the sequence: `« ... "`*string*`" 1 DISP 3 FREEZE HALT ... »`. The stack is available in this state.
- Execute INPUT, which displays a message and a default answer. The program resumes when [ENTER] is pressed. The parameters supplied to INPUT provide considerable control over the appearance of the display and cursor placement. The stack is *not* available in this state, but menus may be changed.
- Executing WAIT with a 0 or −1 parameter, which returns the next keystroke in rc.p format.
- Executing KEY, which returns a key location in rc format, otherwise 0 if no key has been pressed.

Note: *Programs that have been HALTed may be completely terminated by executing KILL.*

A variety of interface options are available by displaying a custom menu before executing the PROMPT, INPUT, or WAIT commands.

A custom menu provides different utility when used in conjunction with the INPUT, PROMPT, or WAIT commands:

- INPUT: provides typing aids.
- PROMPT: can provide execution objects which optionally include CONT to resume program execution.
- WAIT: can provide menu key labels for single keystroke responses, such as menu keys `YES` or `NO`.

Example: INPUT with Custom Menu. The following program fragment (102 bytes, checksum #9067h) accepts a string while providing a menu of common answers. The MENU command at the end of the program restores the previous menu.

```
«
  { "RED" "ORG" "YEL" "GRN" "BLU" "WHT" }
  TMENU "Enter a color code:" "" INPUT 0 MENU
»
```

Example: PROMPT with Custom Menu. The following program (241.5 bytes, checksum #A744h) displays a simple menu which stores zeros or accumulates numbers into variables A and B. When DONE is pressed the CONT command continues the program, which then displays the sums of A and B.

```
«
  { { "CLRA" « 0 'A' STO » }
    { "CLRB" « 0 'B' STO » }
    { "A" « 'A' STO+ » }
    { "B" « 'B' STO+ » }
    ""
    { "DONE" CONT }
  } TMENU
  "Key values into A & B" PROMPT
  A "A" →TAG B "B" →TAG 0 MENU
»
```

Example: WAIT with Custom Menu. The following program fragment (149.5 bytes, checksum #4580h) displays a menu, waits for a YES or NO menu key response, beeps on invalid keys, and returns the keycode of the YES or NO key.

```
« { "YES" "" "" "" "" "NO" } TMENU 0
  DO DROP -1 WAIT UNTIL
    DUP { 11.1 16.1 } SWAP POS
    DUP IF NOT THEN 880 .1 BEEP END
  END 0 MENU
»
```

Recursion

Three conditions must be met to permit recursive programming:

- The system must have an unlimited return stack.
- The system must have an unlimited data stack.
- Programs must be able to call themselves.

The HP 48's data stack and return stack are limited only by available memory, so *recursive programming* is a technique that is available for some forms of problem solving. The programs FIB1 and FIB2 in the HP 48 *Owner's Manual* illustrate that recursion may not always be the fastest technique.

A recursive program uses a technique for repetitive calculation that works by breaking a problem into smaller pieces and calling itself for each piece. A reference manual for the UNIX operating system once defined recursion as follows:

Recursion: See *Recursion*

The definition above is not far off the mark, but it leaves out the test condition for completion.

Factorial Example. The most common illustration of recursive programming is the factorial calculation: $n! = n\times(n-1)\times(n-2)\ldots 2\times 1$, where $1! = 1$. The test for completion is to see if the input parameter $n \leq 1$. The program FACTRL uses recursion:

FACTRL 85.5 Bytes Checksum #BAB7h

```
« → n
  « IF n 1 ≤ THEN 1
    ELSE  n 1 - FACTRL n *
    END
  »
»
```

Quicksort Example. A quicksort works by breaking a list into two smaller lists, then quicksorting each list. The QSORT program below keeps all the items being sorted on the stack, avoiding the overhead associated with building and decomposing list objects. QSORT takes (and returns) the number of stack items to sort from level 1.

The program `« OBJ→ QSORT →LIST »` provides a "front end" to QSORT for list arguments. Large lists should be first stored in a global variable to eliminate excessive overhead in temporary memory processing (see *Temporary Memory*). All the items to be sorted must have the same type, and must be valid arguments to the > command, such as strings or numbers.

QSORT 216 Bytes Checksum #EEF4h

Input: *n-items* *n* →

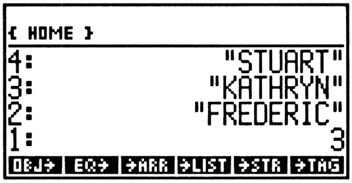
Output: *n-items* *n* →

```
« → n
  « n 2 / ROLL n 3 + 2 n
    START ROT 3 DUPN SWAP ROLLD > - NEXT
    4 - → i
    « n ROLLD i
      IF DUP 1 >
      THEN QSORT
      END
      IF DUP
      THEN 1 SWAP START n ROLLD NEXT i
      END
      n SWAP 1 + -
      IF DUP 1 >
      THEN QSORT
      END DROP n
    »
  »
»
```

Meta–Objects

The term *meta–object* refers to a group of objects and their count that resides on the stack. Since stack operations are by nature very efficient, there are times when decomposing a list onto the stack and performing all operations on the stack will be more efficient than rebuilding the list between operations.

The following display shows a meta–object consisting of three names and their count:



```
{ HOME }
4:                "STUART"
3:               "KATHRYN"
2:              "FREDERIC"
1:                       3
OBJ→ EQ→ →ARR →LIST →STR →TAG
```

The term *meta–stack* refers to a group of objects on the stack, some of which may be meta–objects. The term *position* is used instead of *level* when discussing meta–stacks, because a meta–object actually occupies multiple stack levels.

The following meta–stack consists of the string "FRED" in position 1, and meta–objects in positions 2 and 3:

```
"A" "BB" "C" "DD" 4     21 5 71 3     "FRED"
______________________  ___________   _________  ➞
      Position 3         Position 2   Position 1
```

Notation

To simplify discussions about meta–objects, the following notation is presented. The count is always assumed to be below the elements on the stack.

Stack Notation. The following symbols are used to indicate objects and meta–objects on the stack, where the right–most element is at the bottom of the stack:

< >	An empty meta-object on the stack (which is just a 0, because the meta-object must have a count).
< ... >	An arbitrary meta-object on the stack.
< Obj_1 Obj_2 Obj_3 >	A meta-object composed of three objects.
< ... > Obj	An object in level 1 and a meta-object beginning at level 2.
< Obj ... >	A meta-object on the stack, with Obj at the head. The head is the element farthest from the count. This is equivalent to the decomposition of the list { Obj ... }.
< ... Obj >	A meta-object on the stack, with Obj at the tail. The tail is the element closest to the count. This is equivalent to the decomposition of the list { ... Obj }.
< meta_2 > < meta_1 >	Two meta-objects on the meta-stack.

Utility Names. Several short utility programs are presented below which manipulate meta-objects. The names start with M, for Meta-object, and use the following naming convention:

A	Refers to the addition of an object to a meta-object.
D	Refers to the deletion of an object from a meta-object.
M	Refers to a meta-object.
L	Refers to a list.
H	Refers to the head of a meta-object.
T	Refers to the tail of a meta-object.
Z	Refers to an empty meta-object.
2	Refers to the meta-object in position 2.
→	The phrase "to" (converting *to* another form).

Utilities

To establish an empty meta-object on the stack, just place a zero in level 1. To convert a list or vector into a meta-object, execute OBJ→. To convert a meta-object back to a list, execute →LIST. To convert a meta-object back to a vector, execute →ARRY.

There are many possible routines for meta-object manipulation. The following utility programs are provided to suggest the possibilities. Note that there is no error checking!

MAT adds an object to the tail of a meta-object:

< ... > Obj → < ... Obj >

MAT 25 Bytes Checksum #3538h

```
« SWAP 1 + »
```

MAT2 adds an object to the tail of the second meta-object:

< meta_2 > < meta_1 > Obj → < meta_2 Obj > < meta_1 >

MAT2 53.5 Bytes Checksum #546Eh

```
«
  OVER 3 + ROLLD DUP 2 + ROLL
  1 + OVER 2 + ROLLD
»
```

MAH adds an object to the head of a meta-object:

< ... > Obj → < Obj ... >

MAH 32.5 Bytes Checksum #4F86h

```
« OVER 2 + ROLLD 1 + »
```

MAH2 adds an object to the head of the second meta-object:

$< meta_2 > \quad < meta_1 > \quad Obj \rightarrow < Obj\ meta_2 > \quad < meta_1 >$

MAH2 66 Bytes Checksum #1CACh

```
«
    OVER DUP 4 + PICK + 3 + ROLLD DUP
    2 + ROLL 1 + OVER 2 + ROLLD
»
```

MZ2 places an empty meta-object in meta-stack position 2:

$< meta_1 > \rightarrow < \quad > \quad < meta_1 >$

MZ2 27.5 Bytes Checksum #509Bh

```
« 0 OVER 2 + ROLLD »
```

MDT extracts an element from the tail of a meta-object:

$< \ldots Obj > \rightarrow < \ldots > \quad Obj$

MDT 25 Bytes Checksum #5F4Dh

```
« 1 - SWAP »
```

MDT2 extracts an element from the tail of the second meta-object:

$< Obj_1\ Obj_2\ Obj_3 > \quad < \ldots > \rightarrow < Obj_1\ Obj_2 > \quad < \ldots > \quad Obj_3$

MDT2 56 Bytes Checksum #A95Ch

```
«
  DUP 3 + ROLL OVER
  3 + ROLL 1 - 3 PICK 3 + ROLLD
»
```

MDH extracts an element from the head of a meta-object:

< Obj ... > → < ... > Obj

MDH 32.5 Bytes Checksum #813Dh

```
« 1 - DUP 2 + ROLL »
```

MDH2 extracts an element from the head of the position 2 meta-object:

< Obj_1 Obj_2 Obj_3 > < ... > → < Obj_2 Obj_3 > < ... > Obj_1

MDH2 68.5 Bytes Checksum #BE54h

```
«
  DUP 2 + PICK OVER + 2 + ROLL OVER
  3 + ROLL 1 - 3 PICK 3 + ROLLD
»
```

ML→M converts lists in levels 1 and 2 into meta-objects:

{ list_2 } { list_1 } → < meta_2 > < meta_1 >

ML→M 36 Bytes Checksum #BF3H

```
« SWAP OBJ→ DUP 2 + ROLL OBJ→ »
```

MM→L converts two meta-objects into lists:

< meta_2 > < meta_1 > → { list_2 } { list_1 }

MM→L 36 Bytes Checksum #499Ah

```
« →LIST OVER 2 + ROLLD →LIST SWAP »
```

MAM2 concatenates two meta – objects:

$< meta_1 > \quad < meta_2 > \rightarrow < meta_{1+2} >$

MAM2 31 Bytes Checksum #FAD4h

```
« DUP 2 + ROLL + »
```

MSWAP exchanges two meta – objects:

$< meta_1 > \quad < meta_2 > \rightarrow < meta_2 > \quad < meta_1 >$

MSWAP 73.5 Bytes Checksum #C18Fh

```
«
  DUP 2 + PICK OVER + 2 + → n
  « 1 OVER 1 + START n ROLLD NEXT »
»
```

Using Meta – Objects

Reversing a List. The following program expects a list as input and returns the reversed list as output:

LREV 57.5 Bytes Checksum #D8C1h

```
«
  0 SWAP OBJ→
  DUP 1 SWAP
  START MDT MAT2
  NEXT
  DROP →LIST
»
```

Filtering a List. The following program expects a list as input and returns a list of all string objects in the list in their original order:

SFILT 81 Bytes Checksum #26DBh

```
«
  0 SWAP OBJ→
  DUP 1 SWAP
  START
    MDT IF DUP TYPE 2 SAME
    THEN MAH2
    ELSE DROP
    END
  NEXT
  DROP →LIST
»
```

Searching a Vector. The following program scans an input vector and returns two lists: one with numbers ≤ .5 in level 2, and one with the remaining numbers in level 1:

VSCAN 105.5 Bytes Checksum #3418h

```
«
  0 SWAP OBJ→ OBJ→ DROP
  DUP 1 SWAP
  START MDT
    IF DUP .5 >
    THEN MAH2
    ELSE MAH
    END
  NEXT
  →LIST OVER 2 + ROLLD →LIST
»
```

HP Solve Equation Library

The HP 82211A HP Solve Equation Library application card contains six main applications:

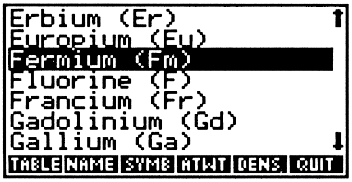
- The Equation Library application contains over 300 equations documented with variable descriptions, units, and pictures.
- The Periodic Table application contains data for 23 properties of 106 elements.
- The Constants Library contains names and values for a collection of physical constants.
- The Finance application provides the Time-Value-of-Money menu from HP financial calculators for compound interest and amortization calculations.
- The Multiple Equation Solver may be used for solving problems that contain more than one equation.
- The Utilities application contains the Minehunt game, several new units, and several new functions used by equations in the Equation Library.

The following pages summarize the applications and provide reference information.

Using Catalogs

The applications in the HP Solve Equation Library use a common environment, called a *catalog*, for viewing and selecting items.

For example, consider the name catalog in the Periodic Table application:



```
Erbium (Er)            ↑
Europium (Eu)
Fermium (Fm)
Fluorine (F)
Francium (Fr)
Gadolinium (Gd)
Gallium (Ga)           ↓
```

The name catalog allows you to choose an element by name. The highlight shows the current item. The arrows on the right side of the display indicate that additional items are available above and/or below the portion of the catalog in the display.

All catalogs provide the following options:

[▲] [▼]	The arrow keys may be used to move the highlight. Press [↰] and an arrow key to move the highlight one screen at a time. Press [↱] and an arrow key to move to the ends of the catalog.
[α]	Press [α] and a letter to move to the next item starting with that letter.
MENU	Menu keys provide various application-specific options.
[ENTER]	Selects the highlighted item. If the item ends with ..., displays the complete item. Press [ATTN] or [ENTER] to return to the catalog.
[ATTN]	Exits the application.

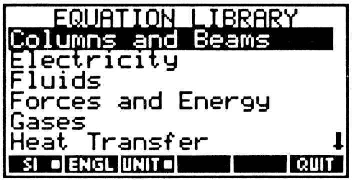
Equation Library

The Equation Library application contains 102 equation titles divided into 15 subject areas. The Equation Library may be used interactively or an equation set may be accessed for use by the solver with the SOLVEQN command.

Interactive Equation Library

The following example illustrates the use of the interactive library. Suppose a projectile is launched at an angle of 35° with an initial velocity of 150 m/s. What is the range of the projectile?

Execute EQNLIB to display the subject catalog:



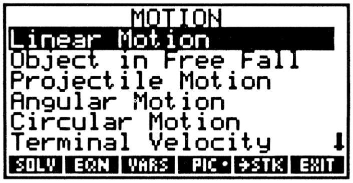
```
    EQUATION LIBRARY
Columns and Beams
Electricity
Fluids
Forces and Energy
Gases
Heat Transfer              ↓
```

When the subject catalog is displayed, you can do the following:

- Select SI or English units by pressing SI or ENG .
- Choose to use or not units by pressing UNIT .
- Press [ENTER] to display the title catalog for the highlighted subject.

If neccessary, press SI and UNIT to place boxes in their menu keys.

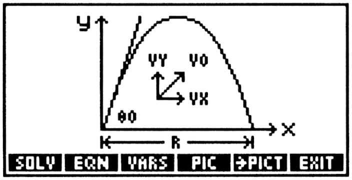
Press [α] [M] [▼] to highlight the MOTION subject, then [ENTER] to display the title catalog:


MOTION
Linear Motion
Object in Free Fall
Projectile Motion
Angular Motion
Circular Motion
Terminal Velocity

The following options are available when you are viewing an equation set:

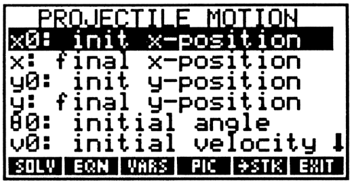
SOLV	Places the current equation set in the solver.
EQN	View the current equation(s) in EquationWriter format.
VARS	Display the variables for the equation set.
PIC	Display the picture associated with the equation set.
→STK	Place the equation set on the stack.
[ENTER]	View the current equation(s) in algebraic format.
EXIT	Return to the subject catalog.

Press [▼] [▼] PIC to display the picture for the Projectile Motion equation set.



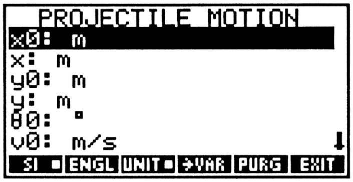
While you are viewing the picture, →PICT may be used to place a copy of the picture in *PICT*.

Press VARS to display the variable catalog:



```
PROJECTILE MOTION
x0: init x-position
x: final x-position
y0: init y-position
y: final y-position
θ0: initial angle
v0: initial velocity ↓
SOLV EQN VARS PIC →STK EXIT
```

Press [NXT] to display the units for each variable:

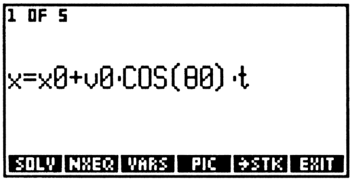


```
PROJECTILE MOTION
x0: m
x: m
y0: m
y: m
θ0: °
v0: m/s                ↓
SI ■ ENGL UNIT■ →VAR PURG EXIT
```

When this page of the variable catalog menu is displayed, the following options are available:

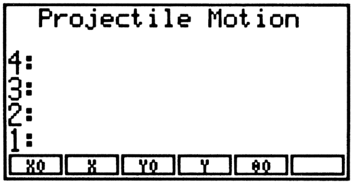
SI	Selects SI units.
ENGL	Selects English units.
UNIT	Selects units or no – units option.
→VAR	Forces the equation set's variables to have the current units.
PURG	Purges the equation set's variables.
EXIT	Returns to the title catalog.
[NXT]	Returns to the first page of the variable menu.

Press [NXT] EQN to display the first of the five equations in the set (NXEQ displays the next equation in the set):

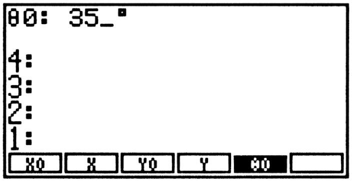


```
1 OF 5

x=x0+v0·COS(θ0)·t

SOLV NXEQ VARS PIC →STK EXIT
```

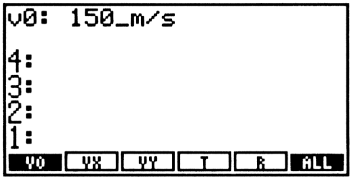
Press SOLV to place the equation set in the multiple equation solver:


```
  Projectile Motion
4:
3:
2:
1:
```

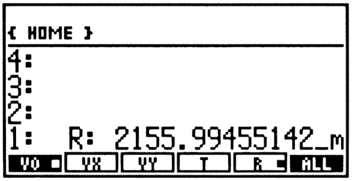
Enter the launch angle by pressing 35 θ0 :


```
θ0:  35_°
4:
3:
2:
1:
```

Notice that the units for the angle are automatically appended to the number you entered. Press NXT to view the next page of variables, and enter the initial velocity by pressing 150 V0 :


```
v0:  150_m/s
4:
3:
2:
1:
```

Solve for the range by pressing ← R :


```
{ HOME }
4:
3:
2:
1:   R: 2155.99455142_m
```

See *Multiple Equation Solver* for a more detailed discussion of the Multiple Equation Solver.

Programmatic Equation Library

The command SOLVEQN may be used to place a set of equations from the Equation Library into the built-in solver for single equations or the Multiple Equation Solver for multiple equation sets. The level 3 and 2 parameters specify the subject and title number. If the level 1 parameter is nonzero, the picture associated with the equation set will be placed in *PICT*.

SOLVEQN	Command
Places Equation Library equation(s) in solver.	
subject title PICT-option →	

The following table shows the subject and title numbers that may be used with the SOLVEQN command. If the *TYPE* is listed as *S*, the title contains a single equation; *M* indicates a set of multiple equations. A *Y* listed under *PICTURE* indicates that a picture is associated with the title.

1	COLUMNS AND BEAMS		
TITLE#	*TITLE*	*TYPE*	*PICTURE*
1	Elastic Buckling	M	Y
2	Eccentric Columns	M	Y
3	Simple Deflection	S	Y
4	Simple Slope	S	Y
5	Simple Moment	S	Y
6	Simple Shear	S	Y
7	Cantilever Deflection	S	Y
8	Cantilever Slope	S	Y
9	Cantilever Moment	S	Y
10	Cantilever Shear	S	Y

2	ELECTRICITY		
TITLE#	*TITLE*	*TYPE*	*PICTURE*
1	Coulomb's Law	S	
2	Ohm's Law and Power	M	
3	Voltage Divider	S	Y
4	Current Divider	S	Y
5	Wire Resistance	S	
6	Series and Parallel R	M	Y
7	Series and Parallel C	M	Y
8	Series and Parallel L	M	Y
9	Capacitive Energy	S	
10	Inductive Energy	S	
11	RLC Current Delay	M	Y
12	DC Capacitor Current	M	
13	Capacitor Charge	S	
14	DC Inductor Voltage	M	
15	RC Transient	S	Y
16	RL Transient	S	Y
17	Resonant Frequency	M	
18	Plate Capacitor	S	Y
19	Cylindrical Capacitor	S	Y
20	Solenoid Inductance	S	Y
21	Toroid Inductance	S	Y
22	Sinusoidal Voltage	M	
23	Sinusoidal Current	M	
3	**FLUIDS**		
1	Pressure at Depth	S	Y
2	Bernoulli Equation	M	Y
3	Flow with Losses	M	Y
4	Flow in Full Pipes	M	Y

4	FORCES AND ENERGY		
TITLE#	*TITLE*	*TYPE*	*PICTURE*
1	Linear Mechanics	M	
2	Angular Mechanics	M	
3	Centripetal Force	M	
4	Hooke's Law	M	Y
5	1D Elastic Collisions	M	Y
6	Drag Force	S	
7	Law of Gravitation	S	
8	Mass – Energy Relation	S	
5	**GASES**		
1	Ideal Gas Law	M	
2	Ideal Gas State Chg	S	
3	Isothermal Expansion	M	
4	Polytropic Processes	M	
5	Isentropic Flow	M	Y
6	Real Gas Law	M	
7	Real Gas State Change	S	
8	Kinetic Theory	M	
6	**HEAT TRANSFER**		
1	Heat Capacity	M	
2	Thermal Expansion	M	Y
3	Conduction	M	Y
4	Convection	M	Y
5	Conduction + Convection	M	Y
6	Black Body Radiation	M	Y
7	**MAGNETISM**		
1	Straight Wire	S	Y
2	Force Between Wires	S	Y
3	B Field in Solenoid	S	Y
4	B Field in Toroid	S	Y

8	MOTION		
TITLE#	*TITLE*	*TYPE*	*PICTURE*
1	Linear Motion	M	
2	Object in Free Fall	M	
3	Projectile Motion	M	Y
4	Angular Motion	M	
5	Circular Motion	M	
6	Terminal Velocity	S	
7	Escape Velocity	S	
9	**OPTICS**		
1	Law of Refraction	S	Y
2	Critical Angle	S	Y
3	Brewster's Law	M	Y
4	Spherical Reflection	M	Y
5	Spherical Refraction	S	Y
6	Thin Lens	M	Y
10	**OSCILLATIONS**		
1	Mass–Spring System	M	Y
2	Simple Pendulum	M	Y
3	Conical Pendulum	M	Y
4	Torsional Pendulum	M	Y
5	Simple Harmonic	M	
11	**PLANE GEOMETRY**		
1	Circle	M	Y
2	Ellipse	M	Y
3	Rectangle	M	Y
4	Regular Polygon	M	Y
5	Circular Ring	M	Y
6	Triangle	M	Y

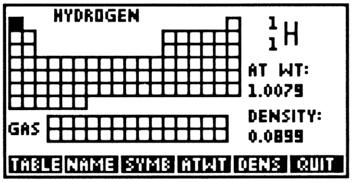
12	SOLID GEOMETRY		
TITLE#	*TITLE*	*TYPE*	*PICTURE*
1	Cone	M	Y
2	Cylinder	M	Y
3	Parallelepiped	M	Y
4	Sphere	M	Y
13	**SOLID STATE DEVICES**		
1	PN Step Junctions	M	Y
2	NMOS Transistors	M	Y
3	Bipolar Transistors	M	Y
4	JFETs	M	Y
14	**STRESS ANALYSIS**		
1	Normal Stress	M	Y
2	Shear Stress	M	Y
3	Stress on an Element	M	Y
4	Mohr's Circle	M	Y
15	**WAVES**		
1	Transverse Waves	M	
2	Longitudinal Waves	M	
3	Sound Waves	M	

Periodic Table

The Periodic Table application contains data for 23 properties of 106 elements. This data may be used in programs to calculate molecular weights of chemical formulas or to display various properties of the elements.

Interactive Periodic Table

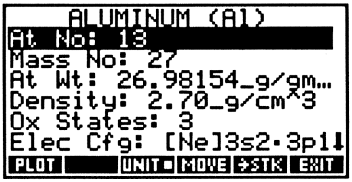
Execute PERTBL to start the interactive periodic table:



When the table is displayed, you can do the following:

- Press the arrow keys to move around the table.
- Use the NAME or SYMB catalogs to locate an element. Use the arrow keys to move the highlight to the desired element, then press TABLE to return to the table or [ENTER] to view the property catalog.
- Press [ENTER] to display the property catalog.
- Press ATWT or DENS to put the atomic weight or density on the stack.
- Press [α] to calculate molecular weights.
- Press QUIT to end the application.

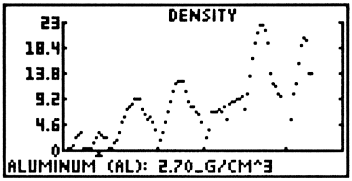
Example: To examine the properties of aluminum, press NAME [▼] [ENTER]:



```
    ALUMINUM (Al)
At No: 13
Mass No: 27
At Wt: 26.98154_g/gm…
Density: 2.70_g/cm^3
Ox States: 3
Elec Cfg: [Ne]3s2·3p1↓
```

Move the highlight to explore the properties of aluminum. Press →STK to return a property to the stack.

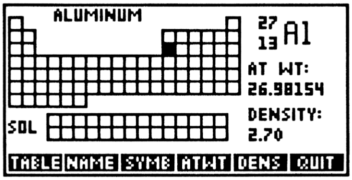
It might be interesting to note the density of aluminum compared to other elements. One way to do this is to plot densities versus atomic number. Move the highlight to Density and press PLOT :



```
ALUMINUM (AL): 2.70_G/CM^3
```

Move the cursor at the bottom of the graph by pressing the arrow keys. Press [ON] to return to the property catalog, or [ENTER] to select a new element.

You can return to the periodic table display by pressing EXIT , and you'll be positioned at aluminum:



```
ALUMINUM
27
13 Al
AT WT:
26.98154
DENSITY:
2.70
```

Calculating Molecular Weights

In the interactive periodic table, press [α], enter the formula, and press [ENTER]. When a formula is being entered, press [↰] [( )] to enter (, or press [↱] [#] to enter). When the result has been displayed, press [ENTER] to return the answer to the stack or [ON] to return to the table.

The MOLWT command may be used in algebraic expressions or programs to calculate the molecular weight of a formula:

MOLWT Function

Calculates molecular weights

'element-name'	→	atomic-weight
'formula'	→	molwt
"formula"	→	molwt
'MOLWT(formula)'		

The string parameter is valid for any formula. If a name parameter represents a valid formula, the molecular weight of that formula will be returned. If a name parameter is not a valid formula, the variable represented by that name will be searched for a formula.

The following table contains examples of valid molecular formulas. The results assume the formula for benzene ("C6H6") is stored in the variable *Benzene*.

Formula	Input	Result
He	He	4.0026_g/gmol
H_2SO_4	H2SO4	98.0734_g/gmol
$Mg(OH)_2$	Mg(OH)2	58.3196_g/gmol
$(CH_3)_2S$	(CH3)2S	62.1294_g/gmol
Benzene	Benzene	78.1134_g/gmol

Extracting Element Data

The PTPROP command may be used in algebraic expressions or programs to return data from the periodic table database. Properties returned as unit objects return real objects if flag 61 is set (no units). Unknown values return the string "–".

PTPROP Function

Returns data from Periodic Table database

atomic – number	property – number	→	data
'element – symbol'	property – number	→	data

'PTPROP(element – symbol,property – number)'

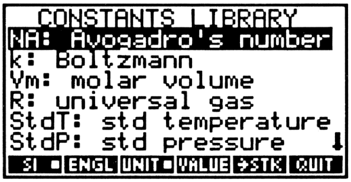
Property	Type	Number
Atomic Number	Real	1
Mass Number	Real	2
Atomic Weight	Unit	3
Density	Unit	4
Oxidation States	String	5
Electronic Configuration	String	6
State	String	7
Melting Point	Unit	8
Boiling Point	Unit	9
Heat of Vaporization	Unit	10
Heat of Fusion	Unit	11
Specific Heat	Unit	12
Group (U.S. Customary)	String	13
Family	String	14
Crystal Structure	String	15
Atomic Volume	Unit	16
Atomic Radius	Unit	17
Covalent Radius	Unit	18
Thermal Conductivity	Unit	19
Electrical Conductivity	Unit	20
First Ionization Potential	Unit	21
Electronegativity (Pauling's)	Unit	22
Oxide Behavior	String	23
Element Name	String	24
Element Symbol	Name	25

Constants Library

The Constants Library contains a collection of names and values of physical constants which may be selected from an interactive catalog or returned using the function CONST.

Constants Catalog

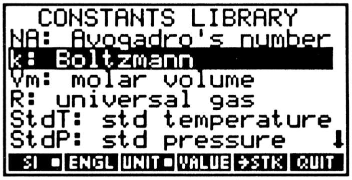
The constants catalog shows the descriptions and values of the constants. Suppose you want to place the SI value of Boltzmann's constant on the stack. Execute CONLIB to display the catalog:



```
   CONSTANTS LIBRARY
NA: Avogadro's number
k: Boltzmann
Vm: molar volume
R: universal gas
StdT: std temperature
StdP: std pressure    ↓
```

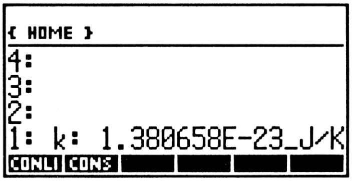
The softkeys `SI`, `ENGL`, and `UNIT` control the type and usage of units. The value returned will respect the SI/English selection regardless of whether units are used.

Press [▼] to highlight Boltzmann's constant, then `VALUE` to display the values instead of the names:



```
   CONSTANTS LIBRARY
NA: Avogadro's number
k: Boltzmann
Vm: molar volume
R: universal gas
StdT: std temperature
StdP: std pressure    ↓
```

Press `→STK` to place the value on the stack, then `QUIT` to exit the application.



```
{ HOME }
4:
3:
2:
1: k: 1.380658E-23_J/K
```

CONST Command

The CONST command may be used in algebraic expressions or programs to return a constant from the Constants Library.

CONST	Function
Returns the value of the specified constant	
name → value	

The units of the value returned are affected by flags 60 (SI if clear, English if set) and 61 (units if clear, no units if set). Note that the value returned respects flag 60 regardless of the state of flag 61.

Example: An equation for free–fall velocity:

```
'V=V0-CONST(g)*T'
```

`CONST(g)` returns the acceleration due to gravity using units as specified by flags 60 and 61.

In a program that performs the same operation, CONST takes the constant's name from the stack:

```
« V0 'g' CONST T * - V STO »
```

Note: Program variables may have the same names as constants if you include ' marks around the constant names so that CONST finds the constant name instead of a variable value.

The table on the following two pages lists the available constants in the Constants Library. Note that one name uses an accented character: ϕ. To type this character, press [α] O [α] [↱][9].

Name	Description
NA	Avogadro's number
k	Boltzmann constant
Vm	Molar volume
R	Universal gas constant
StdT	Standard temperature
StdP	Standard pressure
σ	Stefan – Boltzmann constant
c	Speed of light in vacuum
ε0	Permittivity of vacuum
μ0	Permeability of vacuum
g	Acceleration due to gravity
G	Gravitational constant
h	Planck's constant
hbar	Dirac's constant
q	Electronic charge
me	Electron rest mass
qme	q/me ratio (electron charge – to – mass)
mp	Proton rest mass
mpme	mp/me ratio (proton, electron mass)
α	Fine structure constant
ϕ	Magnetic flux quantum
F	Faraday constant
R∞	Rydberg constant
a0	Bohr radius
μB	Bohr magneton
μN	Nuclear magneton

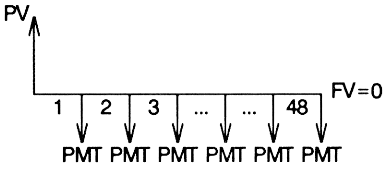
Name	Description
$\lambda0$	Photon wavelength
f0	Photon frequency
λc	Compton wavelength
rad	1 radian
twoπ	2π radians
angl	180° angle (in current trig mode if no units)
c3	Wien's displacement law constant
kq	k/q (Boltzmann, electronic charge)
$\varepsilon0q$	$\varepsilon0/q$ (permittivity, electronic charge)
$q\varepsilon0$	$q \cdot \varepsilon0$ (electronic charge, permittivity)
εsi	Dielectric constant of silicon
εox	Dielectric constant of silicon dioxide
I0	Reference intensity

Finance

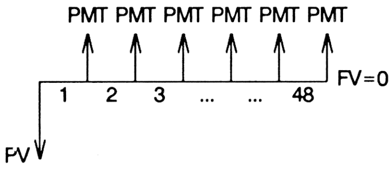
The Finance application may be used for compound interest calculations where identical payments occur over regular periods which coincide with the compounding periods. In Time-Value-of-Money (TVM) calculations money received is displayed as a positive number; money paid out is displayed as a negative number.

Cash Flow Diagrams

TVM cash flow diagrams show money received as an arrow pointing up, and money paid out as an arrow pointing down. The following diagrams illustrate cash flows from the borrower's and lender's point of view:



Loan From Borrower's Point of View



Loan From Lender's Point of View

TVM Calculations

The TVM menu entries store or calculate the following:

N	Number of periods N
I%YR	Annual interest I%YR as a percentage
PV	Present value
PMT	Payment amount
FV	Future value
AMRT	Calculates amortization
	↓ [NXT] ↑
P/YR	Stores the number of payments per year
BEG	Sets Begin mode: payments at each period's start
END	Sets End mode: payments at each period's end

To begin a new TVM problem, set the number of payments per year and Begin or End mode as needed. To change the number of payments per year, key in the new value and press P/YR. Select the payment mode by pressing BEG or END.

To solve TVM problems, enter the values you know and solve for the unknown by pressing ⬅ followed by the appropriate key.

Example: The new 1990 Grande Chrome Deluxe sells for $26,780. The buyer has $8500 for a down payment. Calculate the payments on a four-year loan with 13% annual interest, starting at the ⬅ [LIBRARY] FIN menu in FIX 2 display mode:

Keys:	**Display:**
TVM	12 payments/year END mode
48 N	N: 48.00
13 I%YR	I%YR: 13.00
26780 8500 [−] PV	PV: 18,280.00
0 FV	FV: 0.00
⬅ PMT	1: PMT: -490.41

Amortization

An amortization schedule may be calculated after a loan is specified in the TVM menu by entering the number of periods to amortize and pressing AMRT.

AMORT	Command
Calculates amortization from TVM variables	
payments →	principal interest balance

To continue an amortization, store the balance back into *PV* and execute AMORT for the next number of periods desired.

Amortization Example: A four-year home equity loan of $15,000 has an 11% annual interest rate. Starting in the TVM menu in FIX 2 display mode, calculate the payment, then the interest and principal payment contributions for the first two years:

Keys:	**Display:**
	12 payments/year END mode
48 N	N: 48.00
11 I%YR	I%YR: 11.00
15000 PV	PV: 15,000.00
0 FV	FV: 0.00
⬅ PMT	1: PMT: -387.68
12 AMRT	3: Principal=-3158.24 2: Interest=-1493.92 1: Balance=11841.76
PV 12 AMRT	3: Principal=-3523.71 2: Interest=-1128.45 1: Balance=8318.05

TVMROOT Command

The TVMROOT command may be used in a program to perform TVM calculations.

TVMROOT	Function
Solve for TVM variable using the other TVM variables	
'TVM-variable' → value	

The procedure for programmatic calculations is similar to the keyboard procedure:

- Set the payment mode to begin or end mode using TVMBEG or TVMEND.
- Store the known values in the TVM variables.
- Execute TVMROOT for the unknown variable.

Example: This program returns the amount of money that can be borrowed and the total interest that would be paid given the annual interest rate in level 3, the number of years in level 2, and desired payment in level 1. Remember to supply a negative number for the payment.

AMT 163.5 Bytes Checksum #4B4h

```
«
  TVMEND                 Sets the payment mode
  'PMT' STO              Stores the payment
  12 * 'N' STO           Stores the number of payments
  'I' STO                Stores the annual interest rate
  12 'PYR' STO           Stores the payments per year
  0 'FV' STO             The loan will be paid off
  'PV' TVMROOT           Solves for the loan amount
  DUP 'PV' STO           Stores the present value
  N AMORT                Amortizes the loan
  ROT DROP2              Drops the balance and principal
»
```

Multiple Equation Solver

The Multiple Equation Solver application may be used for solving problems that contain more than one equation.

To use the Multiple Equation Solver, perform the following steps:

- Define the list of equations and store them in *EQ*.
- Execute the MINIT command to establish *Mpar*.
- Execute the MSOLVR command to display the Multiple Equation Solver menu.
- Enter the values for the known variables.
- Solve for any variable or all unknown variables based on the known values:
  - Solve for a single variable by pressing [↰] followed by the appropriate key, or
  - Solve for all the variables by pressing [↰] ALL .
- Review the values for all variables in the menu by pressing [↰] [REVIEW].
- Review the progress catalog by pressing [↱] ALL .

The Multiple Equation Solver menu labels indicate the status of each variable:

Key	Interpretation
X	X unknown
X ■	X unknown, found in the last solution
X	X known, unused in last solution
X ■	X known, used in last solution

Example: Store the equations for the length and volume of a cone (`{ 'L=√(R^2+H^2)' 'V=π*R^2*H/3' }`) in the variable *EQ*, execute MINIT, then MSOLVR. Find the surface area and volume of a right circular cone having a radius of 8 and a height of 24.

Keys:	**Display:**
8 [R]	`R: 8`
24 [H]	`H: 24`
[↰] [ALL]	
[↰] [REVIEW]	`L: 25.2982212813`
	`V: 1608.49543863`
	`R: 8`
	`H: 24`

Programming. The Multiple Equation Solver may be used in programs. The commands MCALC and MUSER may be used to set the unknown and known states of a variable. The command MROOT solves for either a single variable or all unknown variables.

MCALC — Command

Sets Multiple Equation Solver variable to *not* user-defined

'name'	→	
{ name_1 ... name_n }	→	
"ALL"	→	

MROOT — Command

Solves for single or all variables using the Multiple Equation Solver

'name'	→	value
"ALL"	→	

MUSER — Command

Sets Multiple Equation Solver variable to user-defined state

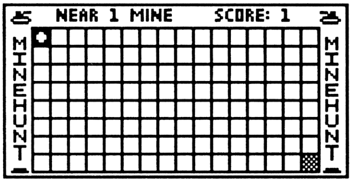
'name'	→	
{ name_1 ... name_n }	→	
"ALL"	→	

Utilities

The Utilities application consists of a game, eight commands, and four new units. The commands and units are described in the next section, *Command Reference*.

Minehunt

The Minehunt game challenges you to navigate a battlefield littered with buried mines. Your mine detector was a low-bid item, and consequently is only able to tell you how many mines are adjacent to your square. You may be beside up to seven mines!



The number keys [2], [8], [4], [6], and arrow keys [▼], [▲], [◀], [▶] move you from square to square. The number keys [1], [3], [7], and [9] permit diagonal movements.

The game ends when you reach the lower-right corner or step on a mine. To interrupt a game when you need to use the HP 48 for other tasks, press [STO]. The state of the game will be stored in *MHpar* until MINEHUNT is executed again.

The score in the upper-right corner tracks the number of squares you have occupied. You may play to either maximize or minimize the number of squares occupied.

The default number of mines is 20. To change this value, store the desired number of mines in the variable *Nmines*. A negative value will show the buried mines.

Command Reference

This command reference lists the stack diagrams for all commands and functions in the HP 82211A HP Solve Equation Library Application Card. Each entry lists the name, description, and stack diagrams if applicable.

NAME Type

Description

Input $\rightarrow$ *Output*

Level$_3$ Level$_2$ Level$_1$ $\rightarrow$ Level$_3$ Level$_2$ Level$_1$

AMORT Command

Calculates amortization from TVM variables

payments $\rightarrow$ principal interest balance

CONLIB Command

Starts the Constants Library

CONST Function

Returns the value of the specified constant

'constname' $\rightarrow$ constant

DARCY Function

Calculates Darcy friction factor

e/D Re $\rightarrow$ d

'symb' x $\rightarrow$ 'DARCY(symb,x)'

x 'symb' $\rightarrow$ 'DARCY(x,symb)'

'symb$_1$' 'symb$_2$' $\rightarrow$ 'DARCY(symb$_1$,symb$_2$)'

---

dB Unit

Dimensionless unit for decibel

ELVERSION Command

Displays the HP 82211A version message

EQNLIB Command

Starts the Equation Library

F0λ Unit

Calculates fraction of black-body emissive power at temperature T between wavelengths 0 and λ

λ T	→	fraction

FANNING Function

Calculates Fanning friction factor

e/D Re	→	f
'symb' x	→	'FANNING(symb,x)'
x 'symb'	→	'FANNING(x,symb)'
'symb$_1$' 'symb$_2$'	→	'FANNING(symb$_1$,symb$_2$)'

gmol Unit

Unit for gram-mole

lbmol Unit

Unit for pound-mole

MINEHUNT Command

Starts the Minehunt game

MINIT Command

Establishes *Mpar* from *EQ*

MITM Command

Changes title and variable menu in *Mpar*

"title" { name$_1$ ... name$_n$ }	→	

MCALC Command

Sets Multiple Equation Solver variable to *not* user-defined state

'name'	→	
{ name$_1$ ... name$_n$ }	→	
"ALL"	→	

MOLWT Function

Calculates molecular weights

'element-name'	→	atomic-weight
'formula'	→	molwt
"formula"	→	molwt
		'MOLWT(formula)'

MROOT Command

Solves for single or all variables using the Multiple Equation Solver

'name'	→	value
"ALL"	→	

MUSER Command

Sets Multiple Equation Solver variable to user-defined state

'name'	→	
{ name_1 … name_n }	→	
"ALL"	→	

MSOLVR Command

Displays the Multiple Equation Solver menu

PERTBL Command

Starts the Periodic Table

PTPROP Function

Returns data from Periodic Table database

atomic-number	property-number	→	data
'element-symbol'	property-number	→	data
'PTPROP(element-symbol,property-number)'			

rpm Unit

Unit for revolutions per minute

SIDENS Function

Intrinsic density of silicon as a function of temperature

T	→	density
'symb'	→	'SIDENS(symb)'

SOLVEQN Command

Places Equation Library equation(s) in solver

subject-number title-number PICT-option →

TDELTA Function

Calculates temperature increment

T_1	T_2	→	increment
'symb'	x	→	'TDELTA(symb,x)'
x	'symb'	→	'TDELTA(x,symb)'
'symb_1'	'symb_2'	→	'TDELTA(symb_1,symb_2)'

Note:

Values returned by TDELTA have level 2 units.

TINC Function

Adds temperature increment

T_1 increment	$\rightarrow$	T_2
'symb' x	$\rightarrow$	'TINC(symb,x)'
x 'symb'	$\rightarrow$	'TINC(x,symb)'
'symb$_1$' 'symb$_2$'	$\rightarrow$	'TINC(symb$_1$,symb$_2$)'

Note:

Values returned by TINC have level 2 units.

TVM Command

Displays the TVM menu

TVMBEG Command

Sets TVM Begin mode

TVMEND Command

Sets TVM End mode

TVMROOT Function

Solve for TVM variable using the other TVM variables

'TVM-variable'	$\rightarrow$	value

ZFACTOR Function

Calculates gas compressibility factor Z

Tr Pr	$\rightarrow$	Z
'symb' x	$\rightarrow$	'ZFACTOR(symb,x)'
x 'symb'	$\rightarrow$	'ZFACTOR(x,symb)'
'symb$_1$' 'symb$_2$'	$\rightarrow$	'ZFACTOR(symb$_1$,symb$_2$)'

Reserved Variables

The applications use reserved variables to store equations and/or state information. These variables may reside in any directory.

Name	Description
MHpar	Saves state of Minehunt game
Mpar	Saves multiple equation solver set
Nmines	Specifies number of Minehunt mines
PTpar	Saves last position in Periodic Table

Flags

The applications use three user flags to control the values and units used in calculations:

Flag	Description	Clear	Set	Default
60	Units Type	SI units	English units	SI units
61	Units Usage	Units used	Units not used	Units used
62	Payment Mode	End mode	Begin mode	End mode

Messages

Hex	Dec	Multiple Equation Solver Messages
10D01	68865	Invalid Mpar
10D02	68866	Single Equation
10D03	68867	EQ Invalid for MINIT
10D04	68868	Too Many Unknowns
10D05	68869	All Variables Known
10D06	68870	Illegal During MROOT
		Finance Messages
10E01	69121	No Solution
10E02	69122	Many or No Solutions
10E03	69123	I%YR/PYR ≤ -100
10E04	69124	Invalid N
10E05	69125	Invalid PYR
10E06	69126	Invalid #Periods
10E07	69127	Undefined TVM Variable
		Constants Library Messages
10F01	69377	Undefined Constant
		Periodic Table Messages
11001	69633	Bad Molecular Formula
11002	69634	Undefined Element
11003	69635	Undefined Property

Library Identifiers

Library	Port 1	Port 2
Equation Library	:1:273	:2:273
Periodic Table	:1:272	:2:272
Constants Library	:1:271	:2:271
Finance Library	:1:270	:2:270
Multiple Equation Solver	:1:269	:2:269
Utilities	:1:268	:2:268
Equation Reference	:1:267	:2:267
Catalog Utility	:1:266	:2:266

System Operations

To invoke a system operation, press and hold [ON], then press and release the second key, then release [ON].

[ON] [A] *and* [F]	Erases all memory (including port 0 and merged memory) and sets the HP 48 to its default states (merged memory remains merged).
[ON] [B]	Cancels the current selection if selected before all keys are released.
[ON] [C]	Brings the calculator back into a known state without resetting user memory. The stack is cleared, the VAR directory is set to HOME, the MTH menu is displayed, User mode is cleared, *PICT* is cleared, and the system configuration is updated to recognize all libraries.
[ON] [D]	Starts the interactive self test (see below).
[ON] [E]	Runs a continuous self test.
[ON] [SPC]	Coma mode: a deep–sleep shutdown which turns off the the system timers (including the clock) and clears the system halt log.
[ON] [PRINT]	Performs a graphics screen dump in HP 82240A/B graphics format (regardless of I/O port selection).
[ON] [+] *or* [-]	Adjusts the display contrast.
[ON] [TIME]	Cancels the next repeating alarm.

System Halt Log

The command WSLOG returns four strings to the stack showing the cause, date, and time of the four most recent system halt events.

The system halt log is not cleared when memory is erased, and may only be cleared by placing the calculator in coma mode.

Example: `3-03/06/90 09:30:10`

This string shows a type three system halt that occurred on the morning of March 6, 1990.

Code	Condition
0	Coma exit
1	Low battery system save
2	I/O timeout
3	Execute through address 0
4	Corrupt time
5	Port change data
7	Hardware difficulty
8	Hardware difficulty
9	Corrupt alarm list
A	Corrupt memory
B	Module pulled
C	Hardware reset
D	Software difficulty
E	Corrupt configuration
F	System RAM card pulled

Note that some events will cause two events to be recorded, and some system halt events will cause a coldstart.

Interactive Self Test

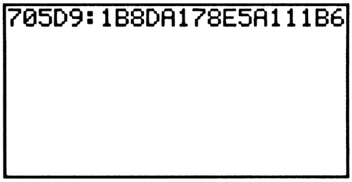
The [ON] [D] sequence enters the HP 48 interactive self test. Once the test has been started, there are a variety of options:

[A]	Displays the CPU speed
[B]	Press [ENTER] for display test patterns
[C]	Internal ROM check
[D]	Internal RAM check
[E]	Keyboard test
[F]	Partial keyboard test
[G]	ESD test monitor. Bars indicate battery status.
[H]	UART loop back test
[I]	Wired UART echo
[J]	Shows what's plugged in
[K]	Test port RAM devices
[L]	Blank display
[M]	Send system time from IR port
[N]	Receive system time from IR port
[O]	Wireless loop back
[P]	Wireless UART echo
[S]	Show test start time
[T]	Show test fail time
[U]	Looping test
[V]	Looping test
[W]	Looping test
[X]	Looping test
[ENTER]	Initialize test times
[Y]	Looping test
[Z]	Looping test
[DEL]	Test summary
[←]	Enters Memory Scanner

Press [ON] [C] to return to the stack display.

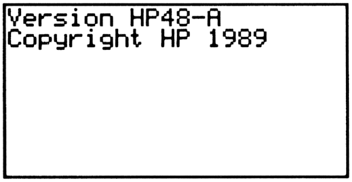
Memory Scanner

The Memory Scanner provides an eight byte window into memory. To start the Memory Scanner, start the interactive self test ([ON] [D]), then press [←]. When finished, press [ON] [C] to return to the stack display.



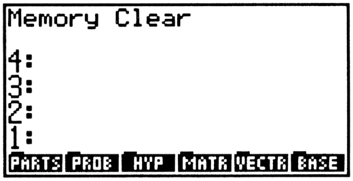
```
705D9:1B8DA178E5A111B6
```

The current address is shown on the left, followed by eight bytes of memory. This address, if executed, shows the revision and copyright message (press [EVAL] to execute at the displayed address):



```
Version HP48-A
Copyright HP 1989
```

Warning: *Pressing* [EVAL] *at any other address than the first address displayed by the Memory Scanner can very likely corrupt memory and produce the following display:*



```
Memory Clear
4:
3:
2:
1:
```

Once the Memory Scanner has been started, there are a variety of options:

[0] – [9]	Hex digit poke
[A] – [F]	Hex digit poke
[+] [−]	Change address by #0001h
[×] [÷]	Change address by #0100h
[▼] [▲]	Change address by #1000h
[ENTER]	Hardware control address
[X]	Display RAM address
[Y]	System Halt Log address
[Z]	Port 1 address
[DEL]	Port 2 address
[EVAL]	Execute starting at this address
[.]	Print current data
[SPC]	Serial memory dump 32K 9600 baud

Printer Control

The following system flags (default clear) control output to the printer as follows:

Flag	Clear	Set
−34	IR printer	Serial printer
−37	Single spaced	Double spaced
−38	Linefeeds	No linefeeds

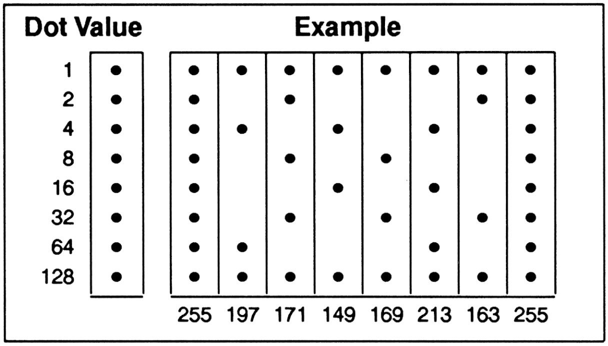
The following control codes guide the operation of the HP 82240B printer:

Printer Command	Control Codes*	
Carriage right	4	
Carriage return/LF	10	
Column graphics	27	n $C_1...C_n$ †
Roman 8 character set ‡	27	248
ISO 8859−1 character set	27	249
Underline off ‡	27	250
Underline on	27	251
Single wide print ‡	27	252
Double wide print	27	253
Self−test	27	254
Reset	27	255

*Decimal value † $1 \leq n \leq 166$ ‡ Default mode

Codes 248 and 249 were not included in the original HP 82240A printer. Characters 148 and 160 were blank on early versions of the HP 82240A printer. The HP 48 character set can be remapped to match the HP 82240A printer with the OLDPRT command.

This example (142.5 bytes, checksum #1380h) prints a simple graphics pattern on the HP 82240.



```
« 27 8                      8 byte graphics command
  255 197 171 149           Graphics data
  169 213 163 255
  " " 1 10 START            Loop start
    SWAP CHR SWAP +         Accumulate data
  NEXT PR1 »                Loop end, print graphics
```

PRTPAR

The reserved variable *PRTPAR* may only reside in the HOME directory. Other variables of the same name in subdirectories will be ignored by the PRINT commands.

PRTPAR →

{ delay "remap" linelen "lineterm" }

Parameter	**Description**	**Default**
delay	Time required to print line: $0 \leq t \leq 6.9$ seconds	1.8
"remap"	Character set remapping string	" "
linelen	Serial print line length	80
"lineterm"	Serial print line terminating characters	" *CR LF* "

Built-In Units

UNIT PREFIXES			
HP 48 Symbol	**Prefix**	**Number**	**Name**
E	exa	+18	quintillion
P	peta	+15	quadrillion
T	tera	+12	trillion
G	giga	+9	billion
M	mega	+6	million
k,K	kilo	+3	thousand
h,H	hecto	+2	hundred
D	deka	+1	ten
d	deci	−1	tenth
c	centi	−2	hundredth
m	milli	−3	thousandth
μ	micro	−6	millionth
n	nano	−9	billionth
p	pico	−12	trillionth
f	femto	−15	quadrillionth
a	atto	−18	quintillionth

Improper prefix-unit combinations that match built-in units are: `au`, `cd`, `ct`, `cu`, `ft`, `flam`, `kph`, `mph`, `min`, `nmi`, `Pa`, `ph`, and `pt`.

DIMENSIONLESS UNITS OF ANGLE		
Unit	**Name**	**Value**
Arcmin	`arcmin`	1/21600 unit circle
Arcsec	`arcs`	1/1296000 unit circle
Degree	`°`	1/360 unit circle
Grad	`grad`	1/400 unit circle
Radian	`r`	$1/2\pi$ unit circle
Steradian	`sr`	$1/4\pi$ unit sphere

Unit	Name	Type	Value
a	Are	area	100 m^2
A	Ampere	electric current	1 A
Å	Angstrom	length	$1\text{x}10^{-10}$ m
acre	Acre	area	4046.87260987 m^2
arcmin	Minute of arc	plane angle	$4.62962962963\text{x}10^{-5}$
arcs	Second of arc	plane angle	$.71604938272\text{x}10^{-7}$
atm	Atmosphere	pressure	$101325 \frac{kg}{m \cdot s^2}$
au	Astronomical unit	length	$1.495979\text{x}10^{11}$ m
b	Barn	area	$1\text{x}10^{-28}$ m^2
bar	Bar	pressure	$100000 \frac{kg}{m \cdot s^2}$
bbl	Barrel	volume	.158987294928 m^3
Bq	Becquerel	activity	$\frac{1}{s}$
Btu	Int'l Table Btu	energy	$1055.05585262 \frac{kg \cdot m^2}{s^2}$
bu	Bushel	volume	.03523907 m^3
c	Speed of light	speed	$299792458 \frac{m}{s}$
C	Coulomb	electric charge	1 A·s
°C	Degree Celsius	temperature	
cal	Calorie	energy	$4.1868 \frac{kg \cdot m^2}{s^2}$
cd	Candela	luminous intensity	1 cd
chain	Chain	length	20.1168402337 m
Ci	Curie	activity	$\frac{3.7\text{x}10^{10}}{s}$
cm	Centimeter	length	.01 m
cm^2	Square centimeter	area	.0001 m^2
cm^3	Cubic centimeter	volume	.000001 m^3
cm/s	Centimeter per second	speed	$.01 \frac{m}{s}$
ct	Carat	mass	.0002 kg

Unit	Name	Type	Value
cu	U.S. cup	volume	$.0002365882365\ \text{m}^3$
d	Day	time	86400 s
dyn	Dyne	force	$.00001\ \frac{\text{kg}\cdot\text{m}}{\text{s}^2}$
erg	Erg	energy	$.0000001\ \frac{\text{kg}\cdot\text{m}^2}{\text{s}^2}$
eV	Electron volt	energy	$1.60219\text{x}10^{-19}\frac{\text{kg}\cdot\text{m}^2}{\text{s}^2}$
F	Farad	capacitance	$1\ \frac{\text{A}^2\cdot\text{s}^4}{\text{kg}\cdot\text{m}^2}$
°F	Degree Fahrenheit	temperature	
fath	Fathom	length	1.82880365761 m
fbm	Board foot	volume	$.002359737216\ \text{m}^3$
fc	Footcandle	illuminance	$.856564774909\ \frac{\text{cd}}{\text{m}^2}$
Fdy	Faraday	electric charge	96487 A·s
fermi	Fermi	length	$1\text{x}10^{-15}$ m
flam	Footlambert	luminance	$3.42625909964\ \frac{\text{cd}}{\text{m}^2}$
ft	Int'l foot	length	.3048 m
ft^2	Square foot	area	$.09290304\ \text{m}^2$
ft^3	Cubic foot	volume	$.028316846592\ \text{m}^3$
ftUS	U.S. survey foot	length	.304800609601 m
ft/s	Feet/second	speed	$.3048\ \frac{\text{m}}{\text{s}}$
ft*lbf	Foot-pound-force	energy	$1.35581794833\ \frac{\text{kg}\cdot\text{m}^2}{\text{s}^2}$
g	Gram	mass	.001 kg
ga	Standard freefall	acceleration	$9.80665\ \frac{\text{m}}{\text{s}^2}$
gal	U.S. gallon	volume	$.003785411784\ \text{m}^3$
galC	Canadian gallon	volume	$.00454609\ \text{m}^3$
galUK	U.K. gallon	volume	$.004546092\ \text{m}^3$

Unit	Name	Type	Value
gf	Gram–force	force	$.00980665 \frac{kg \cdot m}{s^2}$
grad	Grade	plane angle	.0025
grain	Grain	mass	.00006479891 kg
Gy	Gray	absorbed dose	$1 \frac{m^2}{s^2}$
h	Hour	time	3600 s
H	Henry	inductance	$1 \frac{kg \cdot m^2}{A^2 \cdot s^2}$
ha	Hectare	area	$10000 \ m^2$
hp	Horsepower	power	$745.699871582 \frac{kg \cdot m^2}{s^3}$
Hz	Hertz	frequency	$\frac{1}{s}$
in	Inch	length	.0254 m
in^2	Square inch	area	$.00064516 \ m^2$
in^3	Cubic inch	volume	$.000016387064 \ m^3$
inHg	Inch of mercury	pressure	$3386.38815789 \frac{kg}{m \cdot s^2}$
inH2O	Inch of water	pressure	$248.84 \frac{kg}{m \cdot s^2}$
J	Joule	energy	$1 \frac{kg \cdot m^2}{s^2}$
K	Kelvin	temperature	1 K
kcal	Kilocalorie	energy	$4186 \ kg \cdot \frac{m^2}{s^2}$
kg	Kilogram	mass	1 kg
kip	Kilopound–force	force	$4448.22161526 \frac{kg \cdot m}{s^2}$
km	Kilometer	length	1 km
km^2	Square kilometer	area	$1 \ km^2$
knot	Nautical mile per hour	speed	$.514444444444 \frac{m}{s}$
kph	Kilometer per hour	speed	$.277777777778 \frac{m}{s}$

Unit	Name	Type	Value
l	Liter	volume	.001 m^3
lam	Lambert	luminance	3183.09886184 $\frac{cd}{m^2}$
lb	Avoirdupois pound	mass	.45359237 kg
lbf	Pound-force	force	4.44822161526 $\frac{kg \cdot m}{s^2}$
lbt	Troy pound	mass	.3732417 kg
lm	Lumen	luminous flux	$7.95774715459 \times 10^{-2}$ cd
lx	Lux	illuminance	$7.95774715459 \times 10^{-2} \frac{cd}{m^2}$
lyr	Light year	length	$9.46052840488 \times 10^{15}$ m
m	Meter	length	1 m
m^2	Square meter	area	1 m^2
m^3	Cubic meter	volume	1 m^3
μ	Micron	length	.000001 m
MeV	Mega electron volt	energy	$1.60219 \times 10^{-13} \frac{kg \cdot m^2}{s^2}$
mho	Mho	electric conductance	1 $\frac{A^2 \cdot s^3}{kg \cdot m^2}$
mi	Int'l mile	length	1609.344 m
mi^2	Int'l square mile	area	2589988.11034 m^2
mil	Mil	length	.0000254 m
min	Minute	time	60 s
miUS	U.S. statute mile	length	1609.34721869 m
miUS^2	U.S. statute sq. mile	area	258998.47032 m^2
mm	Millimeter	length	.001 m
mmHg	Millimeter of mercury	pressure	133.322368421 $\frac{kg}{m \cdot s^2}$
ml	Milliliter	volume	.000001 m^3
mol	Mole	amount of substance	1 mol
Mpc	Megaparsec	length	$3.08567818585 \times 10^{22}$ m

Unit	Name	Type	Value
mph	Mile per hour	speed	$.44704\ \frac{m}{s}$
m/s	Meter per second	speed	$1\ \frac{m}{s}$
N	Newton	force	$1\ \frac{kg \cdot m}{s^2}$
nmi	Nautical mile	length	1852 m
oz	Ounce	mass	.028349523125 kg
ozfl	U.S. fluid ounce	volume	$2.95735295625 \times 10^{-5}\ m^3$
ozt	Troy ounce	mass	.031103475 kg
ozUK	U.K. fluid ounce	volume	$2.8413075 \times 10^{-5}\ m^3$
P	Poise	dynamic viscosity	$.1\ \frac{kg}{m \cdot s}$
Pa	Pascal	pressure	$1\ \frac{kg}{m \cdot s^2}$
pc	Parsec	length	$3.08567818585 \times 10^{16}\ m$
pdl	Poundal	force	$.138254954376\ \frac{kg \cdot m}{s^2}$
ph	Phot	illuminance	$795.774715459\ \frac{cd}{m^2}$
pk	Peck	volume	$.0088097675\ m^3$
psi	Pound per sq. in.	pressure	$6894.75729317\ \frac{kg}{m \cdot s^2}$
pt	Pint	volume	$.000473176473\ m^3$
qt	Quart	volume	$.000946352946\ m^3$
r	Radian	plane angle	.1591549343092
R	Roentgen	radiation exposure	$.000258\ \frac{A \cdot s}{kg}$
°R	Degree Rankine	temperature	
rad	Rad	absorbed dose	$.01\ \frac{m^2}{s^2}$
rd	Rod	length	5.02921005842 m
rem	Rem	dose equivalent	$.01\ \frac{m^2}{s^2}$
s	Second	time	1 s
S	Siemens	electric conductance	$1\ \frac{A^2 \cdot s^3}{kg \cdot m^2}$

Unit	Name	Type	Value
sb	Stilb	luminance	$10000 \frac{cd}{m^2}$
slug	Slug	mass	14.5939029372 kg
sr	Steradian	solid angle	.0795774715459
st	Stere	volume	$1\ m^3$
St	Stoke	kinematic viscosity	$.0001 \frac{m^2}{s}$
Sv	Sievert	dose equivalent	$.01 \frac{m^2}{s^2}$
t	Metric ton	mass	1000 kg
T	Tesla	magnetic flux	$1 \frac{kg}{A \cdot s^2}$
tbsp	Tablespoon	volume	$1.47867647813 \times 10^{-5}\ m^3$
therm	EEC therm	energy	$105506000 \frac{kg \cdot m^2}{s^2}$
ton	Short ton	mass	907.18474 kg
tonUK	Long (U.K.) ton	mass	1016.0469088 kg
torr	Torr	pressure	$133.322368421 \frac{kg}{m \cdot s^2}$
tsp	Teaspoon	volume	$4.92892159375 \times 10^{-6}\ m^3$
u	Unified atomic mass	mass	1.66057×10^{-27} kg
V	Volt	electrical potential	$1 \frac{kg \cdot m^2}{A \cdot s^3}$
W	Watt	power	$1 \frac{kg \cdot m^2}{s^3}$
Wb	Weber	magnetic flux	$1 \frac{kg \cdot m^2}{A \cdot s^2}$
yd	Int'l yard	length	.9144 m
yd^2	Square yard	area	$.83612736\ m^2$
yd^3	Cubic yard	volume	$.764554857984\ m^3$
yr	Year	time	31556925.9747 s
°	Degree	plane angle	$2.77777777778 \times 10^{-3}$
Ω	Ohm	electric resistance	$1 \frac{kg \cdot m^2}{A^2 \cdot s^3}$

Messages

This program (44 bytes, checksum #7EC6h) retrieves the text of a message given its number by generating the error, then using ERRM to get the text:

```
« → e
  « IFERR e DOERR
    THEN ERRM
    END
» »
```

Hex	Dec	General Messages
001	1	Insufficient Memory
002	2	Directory Recursion
003	3	Undefined Local Name
004	4	Undefined XLIB Name
005	5	Memory Clear
006	6	Power Lost
007	7	Warning:
008	8	Invalid Card Data
009	9	Object In Use
00A	10	Port Not Available
00B	11	No Room in Port
00C	12	Object Not in Port
00D	13	Recovering Memory
00E	14	Try To Recover Memory?
00F	15	Replace RAM, Press ON
010	16	No Mem To Config All
101	257	No Room to Save Stack
102	258	Can't Edit Null Char.
103	259	Invalid User Function
104	260	No Current Equation
106	262	Invalid Syntax

Hex	Dec	Object Types
107	263	Real Number
108	264	Complex Number
109	265	String
10A	266	Real Array
10B	267	Complex Array
10C	268	List
10D	269	Global Name
10E	270	Local Name
10F	271	Program
110	272	Algebraic
111	273	Binary Integer
112	274	Graphic
113	275	Tagged
114	276	Unit
115	277	XLIB Name
116	278	Directory
117	279	Library
118	280	Backup
119	281	Function
11A	282	Command
11B	283	System Binary
11C	284	Long Real
11D	285	Long Complex
11E	286	Linked Array
11F	287	Character
120	288	Code
121	289	Library Data
122	290	External

Hex	Dec	General Messages
123	291	*Null message*
124	292	LAST STACK Disabled
125	293	LAST CMD Disabled
126	294	HALT Not Allowed
127	295	Array
128	296	Wrong Argument Count
129	297	Circular Reference
12A	298	Directory Not Allowed
12B	299	Non-Empty Directory
12C	300	Invalid Definition
12D	301	Missing Library
12E	302	Invalid PPAR
12F	303	Non-Real Result
130	304	Unable to Isolate
Low Memory Messages		
131	305	No Room to Show Stack
132	306	Warning
133	307	Error:
134	308	Purge?
135	309	Out of Memory
136	310	Stack
137	311	Last Stack
138	312	Last Commands
139	313	Key Assignments
13A	314	Alarms
13B	315	Last Arguments
13C	316	Name Conflict
13D	317	Command Line

Hex	Dec	Stack Errors
201	513	Too Few Arguments
202	514	Bad Argument Type
203	515	Bad Argument Value
204	516	Undefined Name
205	517	LASTARG Disabled
EquationWriter Messages		
206	518	Incomplete Subexpression
207	519	Implicit ( ) off
208	520	Implicit ( ) on
Floating-Point Errors		
301	769	Positive Underflow
302	770	Negative Underflow
303	771	Overflow
304	772	Undefined Result
305	773	Infinite Result
Array Messages		
501	1281	Invalid Dimension
502	1282	Invalid Array Element
503	1283	Deleting Row
504	1284	Deleting Column
505	1285	Inserting Row
506	1286	Inserting Column
Statistics Messages		
601	1537	Invalid Σ Data
602	1538	Nonexistent ΣDAT
603	1539	Insufficient Σ Data
604	1540	Invalid ΣPAR
605	1541	Invalid Σ Data LN(Neg)
606	1542	Invalid Σ Data LN(0)

Hex	Dec	Plot/Solve/Stat Messages
607	1543	Invalid EQ
608	1544	Current equation:
609	1545	No current equation.
60A	1546	Enter eqn, press NEW
60B	1547	Name the equation, press ENTER
60C	1548	Select plot type
60D	1549	Empty catalog
60E	1550	undefined
60F	1551	No stat data to plot
610	1552	Autoscaling
611	1553	Solving for
612	1554	No current data. Enter
613	1555	data point, press Σ+
614	1556	Select a model
		Alarm Messages
615	1557	No alarms pending.
616	1558	Press ALRM to create
617	1559	Next alarm:
618	1560	Past due alarm:
619	1561	Acknowledged
61A	1562	Enter alarm, press SET
61B	1563	Select repeat interval
		I/O, Plot, Solve, Stat Messages
61C	1564	I/O setup menu
61D	1565	Plot type:
61E	1566	" "
61F	1567	(OFF SCREEN)
620	1568	Invalid PTYPE
621	1569	Name the stat data, press ENTER
622	1570	Enter value (zoom out if >1), press ENTER

Hex	Dec	I/O, Plot, Solve, Stat
623	1571	Copied to stack
624	1572	x axis zoom w/AUTO.
625	1573	x axis zoom.
626	1574	y axis zoom.
627	1575	x and y-axis zoom.
628	1576	IR/wire:
629	1577	ASCII/binary:
62A	1578	baud:
62B	1579	parity:
62C	1580	checksum type:
62D	1581	translate code:
62E	1582	Enter matrix, then NEW
A01	2561	Bad Guess(es)
A02	2562	Constant?
A03	2563	Interrupted
A04	2564	Root
A05	2565	Sign Reversal
A06	2566	Extremum
Unit Management		
B01	2817	Invalid Unit
B02	2818	Inconsistent Units

Hex	Dec	I/O and Printing
C01	3073	Bad Packet Block Check
C02	3074	Timeout
C03	3075	Receive Error
C04	3076	Receive Buffer Overrun
C05	3077	Parity Error
C06	3078	Transfer Failed
C07	3079	Protocol Error
C08	3080	Invalid Server Cmd.
C09	3081	Port Closed
C0A	3082	Connecting
C0B	3083	Retry #
C0C	3084	Awaiting Server Cmd.
C0D	3085	Sending
C0E	3086	Receiving
C0F	3087	Object Discarded
C10	3088	Packet #
C11	3089	Processing Command
C12	3090	Invalid IOPAR
C13	3091	Invalid PRTPAR
C14	3092	Low Battery
C15	3093	Empty Stack
C16	3094	Row
C17	3095	Invalid Name
	Time Messages	
D01	3329	Invalid Date
D02	3330	Invalid Time
D03	3331	Invalid Repeat
D04	3332	Nonexistent Alarm

Menu Numbers

The commands MENU, TMENU, and RCLMENU store and recall menu numbers in the form *mm.pp*, where *mm* is the menu number and *pp* is the page number.

#	Menu Name	#	Menu Name
0	LAST MENU	30	SOLVE SOLVR
1	CST	31	↰ PLOT
2	VAR	32	PLOT PTYPE
3	MTH	33	PLOT PLOTR
4	MTH PARTS	34	↰ ALGEBRA
5	MTH PROB	35	↰ TIME
6	MTH HYP	36	TIME ADJST
7	MTH MATRX	37	TIME ALRM
8	MTH VECTR	38	TIME ALRM RPT
9	MTH BASE	39	TIME SET
10	PRG	40	↰ STAT
11	PRG STK	41	STAT MODL
12	PRG OBJ	42	↰ UNITS
13	PRG DISP	43	UNITS LENG
14	PRG CTRL	44	UNITS AREA
15	PRG BRCH	45	UNITS VOL
16	PRG TEST	46	UNITS TIME
17	PRINT	47	UNITS SPEED
18	I/O	48	UNITS MASS
19	I/O SETUP	49	UNITS FORCE
20	↰ MODES	50	UNITS ENRG
21	↱ MODES	51	UNITS POWR
22	↰ MEMORY	52	UNITS PRESS
23	↱ MEMORY	53	UNITS TEMP
24	↰ LIBRARY	54	UNITS ELEC
25	LIBRARY PORT 0	55	UNITS ANGL
26	LIBRARY PORT 1	56	UNITS LIGHT
27	LIBRARY PORT 2	57	UNITS RAD
28	↰ EDIT	58	UNITS VISC
29	↰ SOLVE	59	↱ UNITS

Character Codes

NUM	CHR	NUM	CHR	NUM	CHR	NUM	CHR
0	■	32		64	@	96	'
1	■	33	!	65	A	97	a
2	■	34	"	66	B	98	b
3	■	35	#	67	C	99	c
4	■	36	$	68	D	100	d
5	■	37	%	69	E	101	e
6	■	38	&	70	F	102	f
7	■	39	'	71	G	103	g
8	■	40	(	72	H	104	h
9	■	41	)	73	I	105	i
10	■	42	*	74	J	106	j
11	■	43	+	75	K	107	k
12	■	44	,	76	L	108	l
13	■	45	-	77	M	109	m
14	■	46	.	78	N	110	n
15	■	47	/	79	O	111	o
16	■	48	0	80	P	112	p
17	■	49	1	81	Q	113	q
18	■	50	2	82	R	114	r
19	■	51	3	83	S	115	s
20	■	52	4	84	T	116	t
21	■	53	5	85	U	117	u
22	■	54	6	86	V	118	v
23	■	55	7	87	W	119	w
24	■	56	8	88	X	120	x
25	■	57	9	89	Y	121	y
26	■	58	:	90	Z	122	z
27	■	59	;	91	[	123	{
28	■	60	<	92	\	124	|
29	■	61	=	93	]	125	}
30	■	62	>	94	^	126	~
31	...	63	?	95	_	127	▒

NUM	CHR	NUM	CHR	NUM	CHR	NUM	CHR
128	∡	160		192	À	224	à
129	x̄	161	¡	193	Á	225	á
130	∇	162	¢	194	Â	226	â
131	√	163	£	195	Ã	227	ã
132	∫	164	¤	196	Ä	228	ä
133	Σ	165	¥	197	Å	229	å
134	▶	166	¦	198	Æ	230	æ
135	π	167	§	199	Ç	231	ç
136	∂	168	¨	200	È	232	è
137	≤	169	©	201	É	233	é
138	≥	170	ª	202	Ê	234	ê
139	≠	171	«	203	Ë	235	ë
140	α	172	¬	204	Ì	236	ì
141	→	173	–	205	Í	237	í
142	←	174	®	206	Î	238	î
143	↓	175	¯	207	Ï	239	ï
144	↑	176	°	208	Ð	240	ð
145	γ	177	±	209	Ñ	241	ñ
146	δ	178	²	210	Ò	242	ò
147	ε	179	³	211	Ó	243	ó
148	η	180	´	212	Ô	244	ô
149	θ	181	µ	213	Õ	245	õ
150	λ	182	¶	214	Ö	246	ö
151	ρ	183	·	215	×	247	÷
152	σ	184	¸	216	Ø	248	ø
153	τ	185	¹	217	Ù	249	ù
154	ω	186	º	218	Ú	250	ú
155	Δ	187	»	219	Û	251	û
156	Π	188	¼	220	Ü	252	ü
157	Ω	189	½	221	Ý	253	ý
158	■	190	¾	222	Þ	254	þ
159	∞	191	¿	223	ß	255	ÿ

The HP 48 character set can be remapped to match the HP 82240A printer with the OLDPRT command. The character set is based on the ISO 8859 Latin 1 standard, except for characters 127–159.

Object Types

Type	Object	Example
0	Real number	`1.2345`
1	Complex number	`(2.3,4.5)`
2	String	`"ABC"`
3	Real array	`[ 1 2 3 ]`
4	Complex array	`[ (1,2) (3,4) ]`
5	List	`{ "ABC" Var }`
6	Global name	`X`
7	Local name	`y`
8	Program	`« A 2 + »`
9	Algebraic	`'X=Y^2'`
10	Binary integer	`# 247d`
11	Graphics object	`Graphic 131 × 64`
12	Tagged object	`Dist: 34.45`
13	Unit object	`32_ft/s^2`
14	XLIB name	`XLIB 766 1`
15	Directory	`DIR ... END`
16	Library	`Library 766: ...`
17	Backup object	`Backup HOMEDIR`
18	Built-in function	`SIN`
19	Built-in command	`SWAP`
20	System binary	`<2A1h>`
21	Extended real	
22	Extended complex	
23	Linked array	
24	Character	`Character`
25	Code object	`Code`
26	Library Data	`Library Data`
27-31	External object	

Flags

User flags are numbered 1 through 64. System flags are numbered from −1 through −64. By convention, application developers are encouraged to restrict their use of user flags to the range 31 − 64.

All flags are clear by default, execpt for the wordsize (flags −5 → −10).

The related commands SF, CF, FS?, FC?, FS?C, and FC?C are found in the PRG TEST menu. RCLF and STOF return or store a list of two binary integers representing the system and user flag sets.

Flag	Description	Clear	Set	Default
Symbolic Math Flags				
−1	Principal Solution	General solutions	Principal solutions	Clear
−2	Symbolic Constants	Symbolic form	Numeric form	Clear
−3	Numeric Results	Symbolic results	Numeric results	Clear
−4	Not used.			
Binary Integer Math Flags				
−5 → −10	Binary integer wordsize $n+1$: $0 \leq n \leq 63$ Flag −10 is the most significant bit			64
	Binary Integer Base	**−11**	**−12**	DEC
−11, and −12	DEC	Clear	Clear	
	BIN	Clear	Set	
	OCT	Set	Clear	
	HEX	Set	Set	
−13 and −14 are not used.				

Flag	Description	Clear	Set	Default
Coordinate System Flags		**−15**	**−16**	Rect.
−15 and −16	Rectangular	Clear	Clear	
	Cylindrical Polar	Clear	Set	
	Spherical Polar	Set	Set	
Trigonometric Mode Flags		**−17**	**−18**	Degrees
−17 and −18	Degrees	Clear	Clear	
	Radians	Set	Clear	
	Grads	Clear	Set	
Math Exception Flags				
−19	Vector/complex	Vector	Complex	Vector
−20	Underflow Exception	Return 0, set −23 or −24	Error	Clear
−21	Overflow Exception	Return ±MAXR, set −25	Error	Clear
−22	Infinite Result	Error	Return ±MAXR, set −26	Error
−23	Pos. Underflow Ind.	No Exception	Exception	Clear
−24	Neg. Underflow Ind.	No Exception	Exception	Clear
−25	Overflow Indicator	No Exception	Exception	Clear
−26	Infinite Result Ind.	No Exception	Exception	Clear
−27 through −29 are not used.				
Plotting and Graphics Flags				
−30	Function Plotting	*f*(*x*)	*y* and *f*(*x*)	*f*(*x*)
−31	Curve Filling	Filling Enabled	Filling Disabled	Enabled
−32	Graphics Cursor	Visible Light Bkgnd	Visible Dark Bkgnd	Light

Flag	Description	Clear	Set	Default
I/O and Printing Flags				
−33	I/O Device	Serial	IR	Serial
−34	Printing Device	IR	Serial	IR
−35	I/O Data Format	ASCII	Binary	ASCII
−36	RECV Overwrite	New variable	Overwrite	New
−37	Double−Spaced Print	Single	Double	Single
−38	Linefeed	Inserts LF	Suppresses LF	Inserts
−39	Kermit Messages	Msg Displayed	Msg Suppressed	Displayed
Time Management Flags				
−40	Clock Display	TIME menu only	All times	TIME menu
−41	Clock Format	12 hour	24 hour	12 hour
−42	Date Format	MM/DD/YY	DD.MM.YY	MM/DD/YY
−43	Rpt. Alarm Reschedule	Rescheduled	Not Rescheduled	Rescheduled
−44	Acknowledged Alarms	Deleted	Saved	Deleted
Notes: If flag −43 is set, unacknowledged repeat alarms are *not* rescheduled. If flag −44 is set, acknowledged alarms are saved in the alarm catalog.				
Display Format Flags				
−45 → −48	Set the number of digits in Fix, Scientific, and Engineering Modes			0
	Number Display Format	−49	−50	STD
−49 and −50	STD	Clear	Clear	
	FIX	Clear	Set	
	SCI	Set	Clear	
	ENG	Set	Set	
−51	Fraction Mark	Decimal	Comma	Decimal
−52	Single Line Display	Multi−line	Single−line	Multi−line
−53	Precedence	( ) suppressed	( ) displayed	Suppressed

Flag	Description	Clear	Set	Default
Miscellaneous Flags				
−54	Not used.			
−55	Last Arguments	Saved	Not Saved	Saved
−56	Beep	On	Off	On
−57	Alarm Beep	On	Off	On
−58	Verbose Messages	On	Off	On
−59	Fast Catalog Display	Off	On	Off
−60	Alpha Key Action	Twice to lock	Once to lock	Twice
−61	USR Key Action	Twice to lock	Once to lock	Twice
−62	User Mode	Not active	Active	Not active
−63	Vectored Enter	Off	On	Off
−64	Set by GETI or PUTI when their element indices wrap around			

The HP 82211A HP Solve Equation Library application card uses three user flags:

Flag	Description	Clear	Set	Default
60	Units Type	SI units	English units	SI units
61	Units Usage	Units used	Units not used	Units used
62	Payment Mode	End mode	Begin mode	End mode

Subject Index

This index lists the commands and functions in the HP 48, grouped into subject areas. Some commands or functions appear more than once.

BINARY INTEGER MATH

AND	Logical bit-by-bit AND
ASR	Arithmetic shift right
B→R	Binary-to-real conversion
NOT	One's complement
OR	Logical bit-by-bit OR
RCWS	Recalls the binary integer wordsize
RL	Rotates left by one bit
RLB	Rotates left by one byte
RR	Rotates right by one bit
RRB	Rotates right by one byte
R→B	Real-to-binary conversion
SL	Shifts left by one bit
SLB	Shifts left by one byte
SR	Shifts right by one bit
SRB	Shifts right by one byte
STWS	Sets the binary integer wordsize
XOR	Logical bit-by-bit XOR

COMPLEX NUMBER OPERATIONS

ABS	$\sqrt{x^2 + y^2}$
ARG	Returns the polar angle θ of a coordinate pair (x,y)
CONJ	Complex conjugate
C→R	Complex-to-real conversion
i	Symbolic constant *i*
IM	Returns imaginary part of a number or array
NEG	Negates an argument
OBJ→	Complex decomposition
RE	Returns the real part of a complex number
R→C	Real-to-complex conversion
SIGN	Returns unit vector in the direction of the argument
V→	Separates (x,y) into x and y or r and θ
→V2	Combines x and y into (x,y) or (r,θ) if flag −19 is set

ARRAY & LIST OBJECT MANIPULATION

ARRY→	Separate array into individual elements
→ARRY	Combines numbers into an array
CONVERT	Performs a unit conversion
DTAG	Removes all tags from object
EQ→	Separates equation into left and right sides
GET	Gets an element from a list, array, or matrix
GETI	Gets an element from a list, increments and returns the index, and returns the list
LIST→	Separates a list into individual objects
→LIST	Combines objects into a list
OBJ→	Decomposes a composite object into individual components.
POS	Finds an object in a list
PUT	Replaces an element in an array or list
PUTI	Replaces an element in an array or list and increments the index
REPL	Writes an object into another object
SIZE	Finds the number of elements in a list
SUB	Extracts a portion of a list
→TAG	Builds a tagged object
→UNIT	Builds a unit object
→V2	Combines two real numbers into vector
→V3	Combines three real numbers into vector
V→	Separates a 2 or 3 element vector

CONSTANTS

i	Symbolic constant *i*
e	Symbolic constant *e*
MAXR	Symbolic constant – maximum HP 48 real number
MINR	Symbolic constant – minimum HP 48 real number
π	Symbolic constant π

CUSTOMIZATION

ASN	Make a single user-key assignment
DELKEYS	Clears user-key assignments
DEFINE	Creates variable or user-defined function
MENU	Selects a built-in menu or creates a custom menu
ORDER	Rearranges the VAR menu
RCLF	Returns a list containing the system and user flags
RCLKEYS	Lists user-key assignments
RCLMENU	Recalls number and page of active menu
STOF	Sets system and user flags
STOKEYS	Makes multiple user-key assignments
TMENU	Displays temporary built-in or list-defined menu

DATA ENTRY AND EDITING

FREEZE	Freezes up to three display areas
INPUT	Suspends program and waits for data
KEY	Returns key in buffer
LAST	Returns LAST arguments (if saved)
LASTARG	Returns LAST arguments (if saved)
PROMPT	Displays prompt and halts program
TEXT	Selects the stack display
WAIT	Pauses program execution or waits for a key

DEBUGGING AND ERRORS

DOERR	Generates system or user-defined error
ERR0	Clears the last error number
ERRM	Returns the last error message
ERRN	Returns the last error number
HALT	Suspends program execution
IFERR	Begins IFERR test
KILL	Cancels all suspended programs
LAST	Returns arguments (if saved)
LASTARG	Returns arguments (if saved)

DISPLAY MANAGEMENT

CLLCD	Clears the stack display
DISP	Displays an object on line *n*
FREEZE	Freezes up to three display areas
GRAPH	Enters the graphics environment
INPUT	Suspends program and waits for data
PROMPT	Displays prompt and halts program
PVIEW	Displays *PICT* at specified coordinate
TEXT	Selects the stack display

GENERAL MATH

ABS	Absolute value
ARG	Returns the polar angle θ of a coordinate pair (x,y)
CEIL	Next greater integer
CONJ	Complex conjugate
FACT	Factorial or gamma function
FLOOR	Next smaller integer

FP	Fractional part
HMS+	Adds in H.MS format
HMS−	Subtracts in H.MS format
HMS→	Converts a number from H.MS format
→HMS	Converts a number to H.MS format
INV	Inverse (reciprocal)
IP	Integer part
MANT	Returns the mantissa of a number
MAX	Returns the maximum of two numbers
MIN	Returns the minimum of two numbers
MOD	Modulo
NEG	Negates an argument
→Q	Converts number to fractional equivalent
→Qπ	→Q after factoring out π
RE	Returns the real part of a complex number
RND	Rounds fractional part of number
ROOT	Finds a numerical root
RSD	Computes a correction to the solution of a system of equations
R→D	Radians−to−degrees conversion
SIGN	Sign of a number
SQ	Squares a number
TAYLR	Computes a Taylor series approximation
TRNC	Truncates number
XPON	Returns the exponent of a number
XROOT	Returns x^{th} root of y
√	Square root
∫	Integral
∂	Derivative
+	Adds two objects
−	Subtracts two objects
*****	Multiplies two objects
/	Divides two objects
^	Raises a number to a power
%	Percent
%CH	Percent change
%T	Percent total

GRAPHICS AND PLOTTING

ARC	Draws an arc in *PICT*
AUTO	Scales y-axis
AXES	Sets intersection of axes and optionally stores labels
BAR	Selects bar plot
BARPLOT	Draws a bar plot of the data in *ΣDAT*
BLANK	Creates a blank graphics object
BOX	Draws a box in *PICT*
CENTR	Sets center of plot display
CLLCD	Clears the stack display
CONIC	Selects conic plot
C→PX	User-unit to pixel coordinate conversion
DEPND	Specifies plot dependent column, variable, or range
DRAW	Draws a plot
DRAX	Draws axes
ERASE	Erases *PICT*
FUNCTION	Selects function plot
GOR	Superimposes graphics objects
GRAPH	Enters the graphics environment
→GROB	Converts object into graphics object
GXOR	Superimposes and inverts graphics objects
HISTOGRAM	Selects histogram plot
HISTPLOT	Draws a histogram of the data in *ΣDAT*
INDEP	Selects plot independent column, variable or range
LABEL	Labels axes
LCD→	Returns LCD as 131x64 pixel graphics object
→LCD	Displays graphics object
LINE	Draws a line between two coordinates
NEG	Inverts a graphics object
PARAMETRIC	Selects parametric plot
PDIM	Changes the size of *PICT*
PICT	Returns the name *PICT*
PIXOFF	Turns off a pixel in *PICT*
PIXON	Turns on a pixel in *PICT*
PIX?	Tests a pixel in *PICT*
PMAX	Sets the upper-right plot coordinates
PMIN	Sets the lower-left plot coordinates
POLAR	Selects polar plot
PRLCD	Prints an image of the display
PVIEW	Displays *PICT* at specified coordinate

PWRFIT	Selects power curve-fitting model
PX→C	Pixel to user-unit coordinate conversion
RCEQ	Recalls the current equation
REPL	Writes one graphics object into another graphics object
RES	Sets the plot resolution in user unit or pixel intervals
SCALE	Specifies x and y scale in units per 10 pixels
SCATRPLOT	Draws a scatter plot of the data in *ΣDAT*
SCATTER	Selects scatter plot
SIZE	Finds the dimensions of a graphics object
STEQ	Stores into reserved variable *EQ*
SUB	Extracts a sub-grob
TEXT	Displays the stack display
TLINE	Toggles pixels on a straight line
TRUTH	Selects truth plot
XCOL	Specifies *ΣDAT* column as independent variable
YCOL	Specifies a *ΣDAT* column as the dependent variable
YRNG	Specifies y-axis plotting range
***H**	Adjusts the height of a plot
***W**	Adjusts the width of a plot

HYPERBOLIC OPERATIONS

ACOSH	Inverse hyperbolic cosine
ASINH	Inverse hyperbolic sine
ATANH	Inverse hyperbolic tangent
COSH	Hyperbolic cosine
EXPM	Natural exponential minus 1
LNP1	Natural logarithm of (argument + 1)
SINH	Hyperbolic sine
TANH	Hyperbolic tangent

INPUT/OUTPUT AND DATA TRANSFER

BAUD	Sets the baud rate
BEEP	Sounds a beep
BUFLEN	Returns number of characters in the serial buffer
CKSM	Select the checksum scheme
CLOSEIO	Closes the serial port
FINISH	Terminates Kermit server mode
INPUT	Suspends program and waits for data
KERRM	Returns the last Kermit error message

KEY	Returns key in buffer
KGET	Gets named data from a remote device
OPENIO	Opens IR or wired port
PARITY	Sets parity
PKT	Sends commands to server
RECN	Receives and renames file from remote Kermit
RECV	Receives file from remote Kermit, saved in a sender-named object
SBRK	Sends serial break
SEND	Sends object to another Kermit device
SERVER	Selects Kermit Server mode
SRECV	Reads characters from I/O port without Kermit
STIME	Sends serial transmit/receive timeout
TRANSIO	Selects character translation mode
XMIT	Sends string through I/O port without Kermit

LOGARITHMIC OPERATIONS

ALOG	Antilogarithm
e	Symbolic constant *e*
EXP	Natural exponential
EXPM	Natural exponential minus 1
LN	Natural logarithm
LNP1	Natural logarithm of (argument + 1)
LOG	Common (base 10) logarithm
XPON	Returns the exponent of a number

LOGICAL AND RELATIONAL OPERATORS

AND	Logical or binary AND
NOT	Logical or binary NOT
OR	Logical or binary OR
SAME	Tests two objects for equality
XOR	Logical or binary XOR
<	Less-than comparison
≤	Less-than-or-equal comparison
>	Greater-than comparison
≥	Greater-than-or-equal comparison
≠	Not-equal comparison
==	Tests two objects for equality

MATRIX AND ARRAY OPERATIONS

ABS	Square root of sum of squares of elements
ARRY→	Separate array into individual elements
→ARRY	Combines numbers into an array
C→R	Separates complex array into two arrays
CNRM	Column norm
CON	Creates a constant array
CONJ	Complex conjugate
CROSS	Cross product
DET	Determinant of a matrix
DOT	Dot product of two vectors
GET	Gets an element from a list, array, or matrix
GETI	Gets an element from a list, increments and
IDN	Creates an identity matrix
IM	Returns array of imaginary parts from complex array
NEG	Negates elements in an array
PUT	Replaces an element in an array or list
PUTI	Replaces an element in an array or list and increments the index
R→C	Combines two arrays into complex array
RDM	Redimensions an array
RE	Returns array of real parts from complex array
RNRM	Computes row norm of an array
SIZE	Finds the number of elements in an array or matrix
SQ	Squares a matrix
TRN	Transposes a matrix
→V2	Combines two real numbers into vector
→V3	Combines three real numbers into vector
V→	Separates a 2 or 3 element vector

MEMORY MANAGEMENT

ARCHIVE	Makes backup copy of HOME directory
ATTACH	Attaches library to current directory
BYTES	Returns the checksum and number of bytes of an object
CLUSR	Purges all user variables in the current directory
CLVAR	Purges all user variables in the current directory
CRDIR	Creates a directory
DEFINE	Creates user-defined function
DETACH	Detaches library from current directory
FREE	Frees merged memory

HOME	Selects the HOME directory
LIBS	Lists libraries attached to current directory
MEM	Returns available memory
MERGE	Merges RAM card with main memory
NEWOB	Separates object from list or backup name
ORDER	Rearranges the VAR menu
PATH	Returns a list showing the current path
PGDIR	Purges specified directory and its contents
PURGE	Purges one or more variables
PVARS	Returns list of port objects
RCEQ	Recalls the current equation
RCL	Recalls the contents of a variable
RCLF	Returns a list containing the system and user flags
RCLΣ	Recalls the current statistics matrix
RESTORE	Replaces HOME directory with backup copy
SAME	Tests two objects for equality
SIZE	Finds the dimensions of an object
STEQ	Stores into reserved variable *EQ*
STO	Stores an object into a variable
STOΣ	Stores into reserved variable *ΣDAT*
TVARS	Lists the variables of specified type
TYPE	Returns the type of an object
UPDIR	Makes parent directory the current directory
VARS	Returns list of variables in the current directory
VTYPE	Returns type of object in named variable
→	Assigns local variable(s)

MODES AND FLAGS

BIN	Sets binary base
CF	Clears a system or user flag
DEC	Sets decimal base
DEG	Sets Degrees mode
ENG	Sets Engineering display mode
FC?	Tests a system or user flag
FC?C	Tests and clears a system or user flag
FIX	Sets Fix display mode
FS?	Tests a system or user flag
FS?C	Tests and clears a system or user flag
GRAD	Sets Grads mode
HEX	Sets hexadecimal base

OCT	Sets octal base
RAD	Sets Radians mode
RCLF	Returns a list containing the system and user flags
SCI	Sets Scientific display mode
SF	Sets a system or user flag
STD	Sets Standard display mode
STOF	Sets system and user flags

PRINTING

CR	Prints a carriage – right
DELAY	Sets $0 \leq n \leq 6.9$ sec delay between printed lines
OLDPRT	Remaps to HP 82240A character set
PRLCD	Prints an image of the display
PRST	Prints the stack
PRSTC	Prints the stack in compact format
PRVAR	Prints the name and contents of one or more variables
PR1	Prints an object

PROBABILITY

COMB	Combinations of *n* objects taken *r* at a time
FACT	Factorial or gamma function
!	Factorial or gamma function
PERM	Permutations of *n* objects taken *r* at a time
RAND	Returns a random number
RDZ	Sets the random number seed
UTPC	Upper – tail Chi – Square distribuion
UTPF	Upper – tail F – distribution
UTPN	Upper – tail normal distribution
UTPT	Upper – tail t – distribution

PROGRAM BRANCHING AND CONTROL

CASE	Begins CASE structure
CONT	Continues a halted program
DO	Begins DO loop
DOERR	Generates user – defined error
ELSE	Begins ELSE clause
END	Ends program structures
EVAL	Evaluates an object

FOR	Begins FOR loop
HALT	Suspends program execution
IF	Begins IF test
IFERR	Begins IFERR test
IFT	IF ... THEN ... END test
IFTE	IF ... THEN ... ELSE ... END test
INPUT	Suspends program and waits for data
KILL	Cancels all suspended programs
NEXT	Ends FOR ... NEXT or START ... NEXT
→NUM	Evaluates an object to yield a numeric result
OFF	Turns the calculator off
PROMPT	Displays prompt and halts program
REPEAT	Part of WHILE ... REPEAT ... END
START	Begins START ... NEXT or START ... STEP
STEP	Ends FOR ... STEP or START ... STEP
SYSEVAL	Executes a system object
THEN	Begins THEN clause
UNTIL	Part of DO ... UNTIL ... END
UPDIR	Makes parent directory current directory
WAIT	Pauses program execution or waits for a key
WHILE	Begins WHILE ... REPEAT ... END
WSLOG	Returns the four most recent system halts
→	Assigns local variable(s)

STACK MANIPULATION

→ARRY	Combines numbers into an array
CLEAR	Clears the stack
DEPTH	Counts the objects on the stack
DROP	Drops one object from the stack
DROPN	Drops $n+1$ objects from the stack
DROP2	Drops two objects from the stack
DUP	Duplicates one object on the stack
DUPN	Duplicates n objects on the stack
DUP2	Duplicates two objects on the stack
LAST	Returns LAST arguments (if saved)
LASTARG	Returns LAST arguments (if saved)
→LIST	Combines objects into a list
OVER	Copies the object in level 2 into level 1
PICK	Copies nth object into level 1 (excluding n)
ROLL	Moves level $n+1$ object to level 1
ROLLD	Moves the level 2 object to level n
ROT	Moves the level 3 object to level 1
SWAP	Swaps the objects in levels 1 and 2

STATISTICS

BAR	Selects bar plot
BARPLOT	Draws a bar plot of the data in *ΣDAT*
BESTFIT	Executes LR and computes the best curve fit
BINS	Sorts *ΣDAT* data into histogram bins
CLΣ	Purges the statistics matrix
CNRM	Computes the column norm of an array
COLΣ	Specifies dependent and independent columns in *ΣDAT*
CORR	Correlation coefficient
COV	Covariance
EXPFIT	Selects exponential curve-fitting model
HISTOGRAM	Selects histogram plot
HISTPLOT	Draws a histogram of the data in *ΣDAT*
LINFIT	Selects linear curve-fitting model
LOGFIT	Selects logarithmic curve-fitting model
LR	Computes linear regression
MAXΣ	Finds the maximum coordinate values in *ΣDAT*
MEAN	Computes means of the data in *ΣDAT*
MINΣ	Finds the minimum coordinate values in *ΣDAT*
NΣ	Returns the number of data points in *ΣDAT*
PREDV	Predicted dependent variable value
PREDX	Predicted independent variable value
PREDY	Predicted dependent variable value
PWRFIT	Selects power curve-fitting model
RCLΣ	Recalls the current statistics matrix
SCATRPLOT	Draws a scatter plot of the data in *ΣDAT*
SCATTER	Selects scatter plot
SDEV	Computes standard deviations of the data in *ΣDAT*
STOΣ	Stores into reserved variable *ΣDAT*
TOT	Sums the columns in *ΣDAT*
VAR	Computes variances of the data in *ΣDAT*
XCOL	Specifies *ΣDAT* column as the independent variable
YCOL	Specifies a *ΣDAT* column as the dependent variable
XRNG	Specifies x-axis plotting range
Σ	Summation
ΣLINE	Returns best-fit line for data in *ΣDAT*
ΣX	Sum of data in independent *ΣDAT* column
ΣX^2	Sum of squares in independent *ΣDAT* column
ΣY	Sum of data in dependent *ΣDAT* column
ΣY^2	Sum of squares of data in dependent *ΣDAT* column
ΣX*Y	Sum of products in independent and dependent *ΣDAT* columns
Σ+	Appends one or more data points to *ΣDAT*
Σ−	Deletes last row from *ΣDAT*

STRING MANIPULATION

CHR	Makes a one-character string
NUM	Returns character code of a string's first character
POS	Finds a substring in a string
SIZE	Finds the number of characters in a string
STR→	Parses and evaluates a string
→STR	Converts an object to a string
SUB	Extracts a portion of a string

SYMBOLIC MANIPULATION

APPLY	Returns an evaluated expression as the argument to an unevaluated local name
COLCT	Collects like terms
EQ→	Separates equation into left and right sides
EXPAN	Expands an algebraic
e	Symbolic constant *e*
i	Symbolic constant *i*
π	Symbolic constant π
ISOL	Isolates a variable in an equation
↑MATCH	Match-and-replace, beginning with subexpressions
↓MATCH	Match-and-replace, beginning with the top-level expression
→NUM	Evaluates an object to yield a numeric result
OBJ→	Separates outermost function and its arguments
QUAD	Solves a quadratic polynomial
QUOTE	Returns argument expression unevaluated
SHOW	Resolves all references to a name implicit in an algebraic
TAYLR	Computes a Taylor series approximation
∫	Integral
∂	Derivative
|	"Where": appends local name and value to evaluated expression

TRIGONOMETRIC OPERATIONS

ACOS	Arc cosine
ASIN	Arc sine
ATAN	Arc tangent
COS	Cosine
D→R	Degrees-to-radians conversion
R→D	Radians-to-degrees conversion
SIN	Sine
TAN	Tangent

TIME AND ALARMS

ACK	Acknowledges displayed past due alarm
ACKALL	Acknowledges all past due alarms
CLKADJ	Add clock ticks to the system time
DATE	Returns the system date
→DATE	Sets the system date
DATE+	Adds a number of days to a date
DDAYS	Number of days between two dates
DELALARM	Deletes an alarm
FINDALARM	Returns alarm index *n*
HMS+	Adds in H.MS format
HMS−	Subtracts in H.MS format
HMS→	Converts a number from H.MS format
→HMS	Converts a number to H.MS format
RCLALARM	Recalls alarm from alarm list
STIME	Sends serial transmit/receive timeout
STOALARM	Stores alarm in system alarm list
TICKS	Returns time in binary integer clock ticks
TIME	Returns current time as number
→TIME	Sets specified system time
TSTR	Converts date & time numbers to string form

UNIT OBJECT OPERATIONS

CONVERT	Performs a unit conversion
OBJ→	Decomposes a unit object into a number and unit expression
UBASE	Converts unit object to SI base units
UFACT	Factors specified compound unit
→UNIT	Builds a unit object
UVAL	Returns scalar portion of unit object

VARIABLE ARITHMETIC

DECR	Decrements value of specified variable
INCR	Increments and returns value of variable
SCONJ	Conjugates the contents of a variable
SINV	Inverts the contents of a variable
SNEG	Negates the contents of a variable
STO+	Storage arithmetic add
STO−	Storage arithmetic subtract
STO*	Storage arithmetic multiply
STO/	Storage arithmetic divide

Command Reference

This command reference lists the stack diagrams for all commands and functions in the HP 48. Each entry lists the name, characteristics, description, and stack diagrams if applicable.

NAME Characteristics

Description

	Input			*Output*	
Level_3	Level_2	Level_1	$\rightarrow$	Level_3	Level_2	Level_1

Note:
Notes about the function or command

The characteristics are encoded as follows:

Symbol	Characteristic
$\downarrow$	Invertible
∂	Differentiable
$\int$	Integrable

For instance, ACOSH is a function which has an inverse and is differentiable:

ACOSH $\downarrow \partial$ Function

Inverse hyperbolic cosine

z	$\rightarrow$	acosh z
'symb'	$\rightarrow$	'ACOSH(symb)'

The following table lists the terms used in the stack diagrams. Note that system modes may affect the interpretation of input parameters or the results of some functions.

Term	Description
obj	Any object
x or y	Real number
a b c d	Real number
(x,y)	Complex number
z	Real or complex number
m or n	Positive integer real number (rounded if non–integer)
#n or #m	Binary integer
x_unit	Real number with units
x_pa-unit	Real with planar angular units
"string"	Character string
{list}	List of objects
grob	Graphics object
{ #x #y }	Pixel coordinates
hms	Real number in HH.MMSS format
time	Time in HH.MMSS format
repeat	Repeat interval in clock ticks (8192 ticks per second)
date	Date in current MM.DDYYYY or DD.MMYYYY format (flag –42)
T/F	Test result: 0 (false) or non–zero (true)
'symb'	Expression or name treated as an algebraic
[vector]	Real or complex vector
[[matrix]]	Real or complex matrix
[R-array]	Real vector or matrix
[C-array]	Complex vector or matrix
{row col}	Coordinates of an element in a matrix
position	Real number specifying an element in a list, vector, or matrix. May be a list containing two real numbers specifying an element in a matrix.
'name'	Global or local name
'global'	Global name
rc or rc.p	Key location: row–col or row–col.plane (see *User Keys*)
mm.pp	Menu specified as menu.page
d.o.f.	Positive integer degrees of freedom
port	Port number: 0, 1, 2, or & (wildcard)
backup	Backup object
library	Library object
LID	Library identifier (port:library number)

ABS ∂ Function

Absolute value. The absolute value of a vector or matrix is the square root of the sum of squares of the absolute values of the elements.

x	→	$|x|$
(x,y)	→	$\sqrt{x^2 + y^2}$
[vector]	→	|vector|
[[matrix]]	→	|matrix|
'symb'	→	'ABS(symb)'
x_unit	→	|x|_unit

ACK Command

Acknowledges displayed past due alarm

ACKALL Command

Acknowledges all past due alarms

ACOS ↓ ∂ ∫ Function

Arc cosine

z	→	acos z
'symb'	→	'ACOS(symb)'

ACOSH ↓ ∂ Function

Inverse hyperbolic cosine

z	→	acosh z
'symb'	→	'ACOSH(symb)'

ALOG ↓ ∂ ∫ Function

Antilogarithm

z	→	10^z
'symb'	→	'ALOG(symb)'

AND Function

Logical or binary AND

#n_1 #n_2	→	#n_3
x y	→	T/F
x 'symb'	→	'x AND symb'
'symb' x	→	'symb AND x'
'$symb_1$' '$symb_2$'	→	'$symb_1$ AND $symb_2$'
"$string_1$" "$string_2$"	→	"$string_3$"

Note:
String arguments must have the same length

APPLY — ∂ Function

Returns an evaluated expression as the argument to an unevaluated local name

{ $symb_1$ ... $symb_n$ } 'name'	→	'name($symb_1$, ... , $symb_n$)'
		'APPLY(name,$symb_1$, ..., $symb_n$)'

ARC — Command

Draws an arc in *PICT* centered at (x,y), radius *r*, counterclockwise from θ_1 to θ_2

(x,y) *r* θ_1 θ_2	→	
{ #x #y } #r θ_1 θ_2	→	

ARCHIVE — Command

Makes backup copy of HOME directory

:IO: name	→	
:n: name	→	

ARG — ∂ Function

Returns the polar angle θ of a coordinate pair (x,y)

z	→	θ
'symb'	→	'ARG(symb)'

ARRY→ — Command

Separate array into individual elements

[vector]	→	z_1 ... z_n {n}
[[matrix]]	→	z_{11} z_{12} ... z_{nm} {n m}

→ARRY — Command

Combines real or complex numbers into an array

z_1 ... z_n n	→	[vector]
z_{11} z_{12} ... z_{nm} { n m }	→	[[matrix]]

ASIN — ↓ ∂ ∫ Function

Arc sine

z	→	asin z
'symb'	→	'ASIN(symb)'

ASINH — ↓ ∂ Function

Inverse hyperbolic sine

z	→	asinh z
'symb'	→	'ASINH(symb)'

ASN Command

Make a single user–key assignment

object rc.p	→		
'SKEY' rc.p	→		*Reactivates standard key*

ASR Command

Arithmetic shift right (preserves most significant bit)

$\#n_1$	→	$\#n_2$

ATAN ↓ ∂ ∫ Function

Arc tangent

z	→	atan z
'symb'	→	'ATAN(symb)'

ATANH ↓ ∂ Function

Inverse hyperbolic tangent

z	→	atanh z
'symb'	→	'ATANH(symb)'

ATTACH Command

Attaches library to current directory

library–number	→	

AUTO Command

Scales y–axis

AXES Command

Sets intersection of axes and optionally stores labels

(x,y)	→	
{ (x,y) }	→	
{ "Xlabel" "Ylabel" }	→	
{ (x,y) "Xlabel" "Ylabel" }	→	

BAR Command

Selects bar plot

BARPLOT Command

Draws a bar plot of the data in *ΣDAT*

BAUD Command

Sets the serial baud rate: 1200, 2400, 4800, or 9600 (default)

n $\rightarrow$

Note:
The clock should not be displayed during 9600 baud transfers.

BEEP Command

Sounds a beep. Maximum 4400 Hz, 1048 seconds.

Hz secs $\rightarrow$

BESTFIT Command

Selects the statistics model that yields the largest correlation coefficient and executes the LR command

BIN Command

Sets binary base

BINS Command

Sorts the *ΣDAT* data into N bins using the independent variable column as the sort key. The level 1 result shows the number of data points less than and greater than the available bins.

X_{min} width N $\rightarrow$ [[b_1]...[b_N]] [b_L b_R]

BLANK Command

Creates a blank graphics object

#width #height $\rightarrow$ grob

BOX Command

Draws a box in *PICT* with opposite corners defined by user-unit or pixel coordinates

(x,y) (x',y') $\rightarrow$
{ #x #y } { #x' #y' } $\rightarrow$

BUFLEN Command

Returns the number of characters in the serial buffer

$\rightarrow$ n T/F

BYTES Command

Returns the checksum and number of bytes of an object

'global' $\rightarrow$ checksum size
object $\rightarrow$ checksum size

B→R Command

Binary-to-real conversion

#n	→	n

CASE Command

Begins CASE structure

```
        CASE
                test₁ THEN action₁ END
                test₂ THEN action₂ END
                          ...
                testₙ THEN actionₙ END
                default action
        END
```

CEIL Function

Next greater integer

x	→	n
'symb'	→	'CEIL(symb)'
x_unit	→	n_unit

CENTR Command

Sets center of plot display. Supplying x implies (x,0).

(x,y)	→	
x	→	

CF Command

Clears a system or user flag

±n	→	

CHR Command

Makes a one-character string

n	→	"string"

CKSM Command

Select the checksum scheme

n	→	
1		*1-digit arithmetic*
2		*2-digit arithmetic*
3		*3-digit-CRC (default)*

CLEAR Command

Clears the stack

objects	→	

CLKADJ Command

Add clock ticks to the system time (8192 ticks per second)

±ticks	→	

CLLCD Command

Clears the stack display

CLOSEIO Command

Closes the serial port, clears input buffer and KERRM

CLUSR Command
CLVAR

Purges all user variables in the current directory

CLΣ Command

Purges the statistics matrix

CNRM Command

Computes the maximum value of the sums of the absolute values of all elements over all columns

[vector]	→	column-norm
[[matrix]]	→	column-norm

Note:

Since a vector is considered a 1-row matrix, CNRM returns the sum of the absolute values of the elements in the vector.

COLCT Command

Collects like terms

z	→	z
'$symb_1$'	→	'$symb_2$'

COLΣ Command

Specifies dependent and independent columns in *ΣDAT*

independent dependent	→	

COMB Function

Combinations of *n* objects taken *m* at a time

n m	→	$C_{n,m}$
'symb' n	→	'COMB(symb,n)'
n 'symb'	→	'COMB(n,symb)'
'$symb_1$' '$symb_2$'	→	'COMB($symb_1$,$symb_2$)'

CON — Command

Creates a constant array or replaces the contents of an existing array or named array

{rows cols} z	$\rightarrow$	[[matrix]]
[vector_1] z	$\rightarrow$	[vector_2]
[[matrix_1]] z	$\rightarrow$	[[matrix_2]]
'name' z	$\rightarrow$	

CONIC — Command

Selects conic plot

CONJ — ↓ ∂ Function

Complex conjugate

x	$\rightarrow$	x
(x,y)	$\rightarrow$	(x, −y)
[R-array]	$\rightarrow$	[R-array]
[C-array_1]	$\rightarrow$	[C-array_2]
'symb'	$\rightarrow$	'CONJ(symb)'

CONT — Command

Continues a halted program

CONVERT — Command

Performs a unit conversion

x_old y_new	$\rightarrow$	x'_new
x y_pa-unit	$\rightarrow$	x'_pa-unit
x_pa-unit y	$\rightarrow$	x''_pa-unit
x y	$\rightarrow$	x

CORR — Command

Correlation coefficient of *ΣDAT* data in columns specified by COLΣ

	$\rightarrow$	correlation

COS — ↓ ∂ ∫ Function

Cosine

z	$\rightarrow$	cos z
'symb'	$\rightarrow$	'COS(symb)'
x_pa-unit	$\rightarrow$	cos x

COSH ↓∂∫ Function

Hyperbolic cosine

z	→	cosh z
'symb'	→	'COSH(symb)'

COV Command

Covariance of ΣDAT data in columns specified by COLΣ

	→	covariance

CR Command

Prints a carriage-right

CRDIR Command

Creates a directory

'name'	→	

CROSS Command

Cross product

[A] [B]	→	[A × B]

C→PX Command

User-unit to pixel coordinate conversion

(x,y)	→	{ #col #row }

C→R Command

Complex-to-real conversion

(x,y)	→	x y
[C-array]	→	[R-array$_{real}$] [R-array$_{imag}$]

DATE Command

Returns the system date

	→	date

→DATE Command

Sets the system date

date	→	

DATE+ Command

Adds a number of days to a date

date #days	→	date'

DDAYS Command

Number of days between two dates

date$_1$ date$_2$	→	Δdays

DEC Command

Sets decimal base

DECR Command

Decrements value of specified variable

'name'	→	x

DEFINE Command

Creates user-defined function

'equation'	→	

DEG Command

Sets Degrees mode

DELALARM Command

Deletes one alarm or all alarms from the system alarm list

n	→	*Deletes specified alarm*
0	→	*Deletes all alarms*

DELAY Command

Sets $0 \leq n \leq 6.9$ second delay between printed lines (1.8 second default)

n	→	

DELKEYS Command

Clears user-key assignments

rc.p	→	*Clears a single key*
{ rc.p_1 rc.p_2 ... }	→	*Clears a list of keys*
S	→	*Clears standard key definitions*
{ S rc.p_1 rc.p_2 ... }	→	*Clears list of keys & std key defs*
0	→	*Clears all user keys*

DEPND Command

Specifies plot dependent column, variable, or range

n	→	
'name'	→	
{ name }	→	
start end	→	
{ start end }	→	
{ name start end }	→	

DEPTH Command

Counts the objects on the stack

objects	→	objects n

DET Command

Determinant of a square matrix

[[matrix]] → determinant

DETACH Command

Detaches library from current directory

library-number →

DISP Command

Displays an object in medium font (5x7) on line *n*, where *n*=1 is the top line, *n*=7 is the bottom line

object n →

DO Command

Begins DO loop

DO loop-clause **UNTIL** test-clause **END**

DOERR Command

Generates system or user-defined error

0	→	*Simulates* [ATTN]
n	→	*Issues machine error* n
#n	→	*Issues machine error* n
"string"	→	*Issues string error*

DOT Command

Dot product of two vectors

[**A**] [**B**] → x

DRAW Command

Draws a plot

DRAX Command

Draws axes

DROP Command

Drops one object from the stack

object →

DROPN Command

Drops *n* and *n* objects from the stack

obj_n … obj_1 n →

DROP2 Command

Drops two objects from the stack

$obj_2 \quad obj_1 \quad \rightarrow$

DTAG Command

Removes all tags from object

:tag:obj → obj

DUP Command

Duplicates one object on the stack

obj → obj obj

DUPN Command

Duplicates *n* objects on the stack (excluding *n*)

$obj_n \; \ldots \; obj_1 \quad n \quad \rightarrow \quad obj_n \; \ldots \; obj_1 \quad obj_n \; \ldots \; obj_1$

DUP2 Command

Duplicates two objects on the stack

$obj_1 \quad obj_2 \quad \rightarrow \quad obj_1 \quad obj_2 \quad obj_1 \quad obj_2$

D→R Function

Degrees-to-radians conversion

x → (π/180)x

'symb' → 'D→R(symb)'

e ∫ Function

Symbolic constant *e*

→ 2.71828182846

ELSE Command

Begins false-clause in IF ... THEN ... ELSE ... END
or IFERR ... THEN ... ELSE ... END

END Command

Ends program structures

ENG Command

Sets Engineering display mode

n →

EQ→ Command

Separates equation into left and right sides

'symb$_1$=symb$_2$'	→	'symb$_1$' 'symb$_2$'
z	→	z 0
'name'	→	'name' 0
x_unit	→	x_unit 0

ERASE Command

Erases *PICT*

ERRM Command

Returns the last error message

	→	"error message"

ERR0 Command

Clears the last error number

ERRN Command

Returns the last error number

	→	#n

EVAL Command

Evaluates an object

obj	→	
:port:name	→	
:port:{path name}	→	
{ port:name$_1$ port:name$_2$ ... }	→	

EXP ↓ ∂ ∫ Function

Natural exponential

z	→	exp z
'symb'	→	'EXP(symb)'

EXPAN Command

Expands an algebraic

z	→	z
'symb$_1$'	→	'symb$_2$'

EXPFIT Command

Selects exponential curve–fitting model

EXPM ↓ ∂ ∫ Function

Natural exponential minus 1

x	→	exp(x) – 1
'symb'	→	'EXPM(symb)'

FACT Function

Factorial or gamma function

n	→	n!
x	→	$\Gamma(x+1)$
'symb'	→	'FACT(symb)'

FC? Command

Tests a system or user flag

±n	→	T/F

FC?C Command

Tests and clears a system or user flag

±n	→	T/F

FINDALARM Command

Returns alarm index *n*

First alarm due after a date and time:

{ date time }	→	n

First alarm due on a specified date:

date	→	n

First past due alarm:

0	→	n

FINISH Command

Terminates Kermit server mode.

FIX Command

Sets Fix display mode

n	→	

FLOOR Function

Next smaller integer

x	→	n
'symb'	→	'FLOOR(symb)'
x_unit	→	n_unit

FOR Command

Begins FOR loop

start end **FOR** counter loop-clause **NEXT**
start end **FOR** counter loop-clause increment **STEP**

FP Function

Fractional part

	x	→	y
	'symb'	→	'FP(symb)'
	x_unit	→	y_unit

FREE Command

Frees merged memory

LID	port	→	
{ }	port	→	
	:port:name	→	
{ :port:names ... LIDs }	port	→	

FREEZE Command

Freezes up to three display areas. The least significant bits control which area will be frozen.

n	→	
Bit: 0		*Status area*
1		*Stack & command line*
2		*Menu area*

FS? Command

Tests a system or user flag

±n	→	T/F

FS?C

Tests and clears a system or user flag

±n	→	T/F

FUNCTION Command

Selects function plot

GET Command

Gets an element from a list, vector, or matrix

{ list }	position	→	object
'name'	position	→	object
[vector]	position	→	z
[[matrix]]	position	→	z
[[matrix]]	{ row col }	→	z
'name'	{ row col }	→	z

GETI

Command

Gets an element from a list, increments and returns the position, and returns the list

{ list }	position	→	{ list } position' object
'name'	position	→	'name' position' object
[vector]	position	→	[vector] position' z
[[matrix]]	position	→	[[matrix]] position' z
[[matrix]]	{ row col }	→	[[matrix]] { row col' } z
'name'	{ row col }	→	'name' { row col' } z

GOR

Command

Superimposes grob' onto grob at the specified coordinates

grob	(x,y)	grob'	→	grob"
grob	{ #x #y }	grob'	→	grob"
PICT	(x,y)	grob'	→	
PICT	{ #x #y }	grob'	→	

GRAD

Command

Sets Grads mode

GRAPH

Command

Enters the Graphics environment until [ATTN] is pressed

→GROB

Command

Converts object into graphics object

object	n	→	grob
	0		*EquationWriter picture*
	1		*Small font* (*3x5*)
	2		*Medium font* (*5x7*)
	3		*Large font* (*5x9*)

GXOR

Command

Superimposes and inverts grob' onto grob at the specified coordinates

grob	(x,y)	grob'	→	grob"
grob	{ #x #y }	grob'	→	grob"
PICT	(x,y)	grob'	→	
PICT	{ #x #y }	grob'	→	

HALT Command

Suspends program execution

HEX Command

Sets hexadecimal base

HISTOGRAM Command

Selects histogram plot

HISTPLOT Command

Draws a histogram of the data in *ΣDAT*

HMS+ Command

Adds in H.MS format

$hms_1 \quad hms_2 \rightarrow hms_1 + hms_2$

HMS- Command

Subtracts in H.MS format

$hms_1 \quad hms_2 \rightarrow hms_1 - hms_2$

HMS→ Command

Converts a number from H.MS format

hms → x

→HMS Command

Converts a number to H.MS format

x → hms

HOME Command

Selects the HOME directory

i ∂ Function

Symbolic constant *i*

→ (0,1)

IDN Command

Creates an identity matrix

n	→	[[n x n real-identity-matrix]]
[[matrix]]	→	[[identity-matrix]]
'name'	→	*replaces named matrix*

IF — Command

Begins IF test

IF test **THEN** true – clause **END**

IF test **THEN** true – clause **ELSE** false – clause **END**

IFERR — Command

Begins IFERR test

IFERR test **THEN** true – clause **END**

IFERR test **THEN** true – clause **ELSE** false – clause **END**

IFT — Command

IF ... THEN ... END test. Executes *object* if *T/F* is true.

```
T/F  object  →
```

IFTE — ∂ Function

IF ... THEN ... ELSE ... END test. Executes *true – obj* if *T/F* is true, otherwise executes *false – obj*.

```
   T/F  true–obj  false–obj  →
'symb'  true–obj  false–obj  →
                 'IFTE(symb,true–obj,false–obj)'
```

IM — Function

Returns imaginary part of a number or array

```
        x  →  0
    (x,y)  →  y
[R-array]  →  [zero R-array]
[C-array]  →  [R-array]
   'symb'  →  'IM(symb)'
```

INCR — Command

Increments and returns value of variable

```
'name'  →  x
```

INDEP — Command

Specifies plot independent column, variable or range

```
                    n  →
               'name'  →
             { name }  →
            start end  →
        { start end }  →
   { name start end }  →
```

INPUT

Command

Suspends program, displays message, and waits for data. *mode* can be ALG, α, or V. The level 1 list may contain any of the options in any order.

"message" "prompt"	→	"result"
"message" { "prompt" column mode }	→	"result"
"message" { "prompt" { row col } mode }	→	"result"

INV

↓ ∂ ∫ Function

Inverse (reciprocal)

z	→	1/z
[[matrix]]	→	[[1/matrix]]
'symb'	→	'INV(symb)'
x_unit	→	1/x_1/unit

IP

Function

Integer part

x	→	n
'symb'	→	'IP(symb)'
x_unit	→	n_unit

ISOL

Command

Isolates a variable in an equation

'symb$_1$' 'global'	→	'symb$_2$'

KERRM

Command

Returns the last Kermit error message

	→	"message"

KEY

Command

Returns 0 if no key in has been pressed, otherwise 1 in level 1 and the keycode in level 2.

	→	0
	→	rc 1

KGET

Command

Gets named data from a remote device

'name'	→
"name"	→
{ remote-name local-name }	→
{ name$_1$ name$_2$ ... }	→
{ { remote-name$_1$ local-name$_1$ } name$_2$ ... }	→

KILL Command

Cancels all suspended programs

LABEL Command

Labels axes

LAST
LASTARG Command

Returns arguments (saved if flag −55 is clear)

→ *Last-Argument*(s)

LCD→ Command

Returns LCD as 131x64 pixel graphics object

→ grob

→LCD Command

Displays graphics object at the upper-left corner of the display

grob →

LIBS Command

Lists library objects attached to current directory

→ { "title_1" library-number_1 port_1 ... }

LINE Command

Draws a line between two coordinates

(x,y) (x',y') →

{ $\#x_1$ $\#y_1$ } { $\#x_2$ $\#y_2$ } →

LINFIT Command

Selects linear curve-fitting model

LIST→ Command

Separates a list into individual objects

{ obj_1 ... obj_n } → obj_1 ... obj_n n

→LIST Command

Combines objects into a list

obj_1 ... obj_n n → { obj_1 ... obj_n }

LN ↓ ∂ ∫ Function

Natural logarithm

z → ln z

'symb' → 'LN(symb)'

LNP1	↓∂ Function
Natural logarithm of (argument + 1)	

x	→	ln(1+x)
'symb'	→	'LNP1(symb)'

LOG	↓∂∫ Function
Common (base 10) logarithm	

z	→	log z
'symb'	→	'LOG(symb)'

LOGFIT	Command
Selects logarithmic curve-fitting model	

LR	Command
Computes linear regression of *ΣDAT* data	

	→	intercept slope

MANT	Function
Returns the mantissa of a number	

x	→	y
'symb'	→	'MANT(symb)'

↑MATCH	Command
Match-and-replace, beginning with subexpressions	

'symb'	{ 'pattern' 'replacement' }	→	'result' T/F
'symb'	{ 'pat' 'repl' 'conditional' }	→	'result' T/F

↓MATCH	Command
Match-and-replace, beginning with the top-level expression	

'symb'	{ 'pattern' 'replacement' }	→	'result' T/F
'symb'	{ 'pat' 'repl' 'conditional' }	→	'result' T/F

MAX	Function
Returns the maximum of two numbers	

x y	→	max(x,y)
x 'symb'	→	'MAX(x,symb)'
'symb' x	→	'MAX(symb,x)'
'$symb_1$' '$symb_2$'	→	'MAX($symb_1$,$symb_2$)'
x y_pa-unit	→	max(x,UBASE(y))
x_pa-unit y	→	max(UBASE(x),y)
x_unit y_unit	→	max(x,y)_unit

MAXR — ∂ Function

Symbolic constant – maximum HP 48 real number

	→	9.99999999999E499

MAXΣ — Command

Finds the maximum column values of the data in *ΣDAT*

	→	x
	→	[x_1 ... x_m]

MEAN — Command

Computes means of the data in *ΣDAT*

	→	x
	→	[x_1 ... x_m]

MEM — Command

Returns available memory

	→	x

MENU — Command

Selects a built-in menu or creates a custom menu (see *Menus*)

mm.pp	→	
'list-name'	→	
{ *names and commands* }	→	

MERGE — Command

Merges RAM card with main memory

port	→	

MIN — Function

Returns the minimum of two numbers

x y	→	min(x,y)
x 'symb'	→	'MIN(x,symb)'
'symb' x	→	'MIN(symb,x)'
'$symb_1$' '$symb_2$'	→	'MIN($symb_1$,$symb_2$)'
x y_pa-unit	→	min(x,UBASE(y))
x_pa-unit y	→	min(UBASE(x),y)
x_unit y_unit	→	min(x,y)_unit

MINR ∂ Function

Symbolic constant – minimum HP 48 real number

	→	1.E−499

MINΣ Command

Finds the minimum column values of the data in *ΣDAT*

	→	x
	→	[x_1 … x_m]

MOD Function

Modulo

x y	→	x mod y
x 'symb'	→	'MOD(x,symb)'
'symb' x	→	'MOD(symb,x)'
'$symb_1$' '$symb_2$'	→	'MOD($symb_1$,$symb_2$)'

NEG ↓∂ Function

Negates an argument

z	→	−z
#n_1	→	#n_2 (*two's complement*)
x_unit	→	−x_unit
[vector]	→	[−vector]
[[matrix]]	→	[[−matrix]]
'symb'	→	'−(symb)'
grob	→	inverted−grob
PICT	→	*inverts PICT*

NEWOB Command

Separates object from list or backup object (see *Temporary Memory*)

object	→	object

NEXT Command

Ends FOR ... NEXT or START ... NEXT

NOT Function

Logical or binary NOT

#n_1	→	#n_2
x	→	T/F
'symb'	→	'NOT(symb)'
"$string_1$"	→	"$string_2$"

NUM Command

Returns character code of a string's first character

"string"	$\rightarrow$	n

→NUM Command

Evaluates an object to yield a numeric result

object	$\rightarrow$	z

NΣ Command

Returns the number of data points in Σ*DAT*

	$\rightarrow$	n

OBJ→ Command

Decomposes a composite object into individual components. String objects are executed as a command line after the " " delimiters have been removed.

:tag:object	$\rightarrow$	object "tag"
(x,y)	$\rightarrow$	x y
x_units	$\rightarrow$	x 1_units
'X + Y'	$\rightarrow$	'X' 'Y' 2 +
[x_1 ... x_n]	$\rightarrow$	x_1 ... x_n n
[[x_{11} x_{12} ... x_{nm}]]	$\rightarrow$	x_1 ... x_n { n m }
{ obj_1 ... obj_n }	$\rightarrow$	obj_1 ... obj_n n
"string"	$\rightarrow$	

OCT Command

Sets octal base

OFF Command

Turns the calculator off

OLDPRT Command

Remaps printer output to the HP 82240A character set

OPENIO Command

Opens IR or wired port

OR — Command

Logical or binary OR

#n_1 #n_2	→	#n_3
x y	→	T/F
x 'symb'	→	'x OR symb'
'symb' x	→	'symb OR x'
'symb_1' 'symb_2'	→	'symb_1 OR symb_2'
"string_1" "string_2"	→	"string_3"

Note:

String arguments must have the same length

ORDER — Command

Rearranges the VAR menu

{ names }	→	

OVER — Command

Copies the object in level 2 into level 1

obj_2 obj_1	→	obj_2 obj_1 obj_2

PARAMETRIC — Command

Selects parametric plot

PARITY — Command

Sets parity. *$n<0$* indicates transmit parity only.

n	→	
0		*none*
1		*odd*
2		*even*
3		*mark*
4		*space*

PATH — Command

Returns a list showing the current path

	→	{ HOME directory−names }

PDIM — Command

Changes the size of *PICT*

(x_{min},y_{min}) (x_{max},y_{max})	→	*Changes PICT relative to the current user coordinates*
#horizontal #vertical	→	*Does not affect current user coordinates*

PERM Function

Permutations of *n* objects taken *m* at a time

n m	→	$P_{n,m}$
'symb' n	→	'PERM(symb,n)'
n 'symb'	→	'PERM(n,symb)'
$'symb_1'$ $'symb_2'$	→	$'PERM(symb_1,symb_2)'$

PGDIR Command

Purges specified directory and its contents

'name'	→	

PICK Command

Copies *n*th object into level 1 (excluding *n*)

obj_n ... obj_1 n	→	obj_n ... obj_1 obj_n

PICT Command

Returns the name *PICT* to level 1

	→	*PICT*

PIXOFF Command

Turns off a pixel in *PICT*

(x,y)	→	
{ #x #y }	→	

PIXON Command

Turns on a pixel in *PICT*

(x,y)	→	
{ #x #y }	→	

PIX? Command

Tests a pixel in *PICT*

(x,y)	→	T/F
{ #x #y }	→	T/F

PKT Command

Sends commands to server

"contents" "type"	→	"response"

PMAX Command

Sets the upper-right plot coordinates

(x,y) →

PMIN Command

Sets the lower-left plot coordinates

(x,y) →

POLAR Command

Selects polar plot

POS Command

Finds a substring in a string or finds an object in a list

"string" "substring" → n

{ list } obj → n

PREDV Command

Predicted dependent variable value

x → predicted-value

PREDX Command

Predicted independent variable value

y → predicted-value

PREDY Command

Predicted dependent variable value

x → predicted-value

PRLCD Command

Prints an image of the display

PROMPT Command

Displays prompt and halts program

"prompt" →

PRST Command

Prints the stack

PRSTC Command

Prints the stack in compact format

PRVAR — Command

Prints the name and contents of one or more variables

'name'	→	
:port:name	→	
{ name$_1$ name$_2$ ... }	→	

PR1 — Command

Prints the level 1 object

object	→	object

PURGE — Command

Purges one or more variables

'global'	→	
{ global$_1$ global$_2$ ... }	→	
{ port:name$_1$ port:name$_2$ ... }	→	
:port:name	→	
LID	→	
PICT	→	

PUT — Command

Replaces an element in an array or list

{ list$_1$ } position obj	→	{ list$_2$ }
'name' position obj	→	
[vector$_1$] position z	→	[vector$_2$]
[[matrix$_1$]] position z	→	[[matrix$_2$]]
[[matrix$_1$]] { row col } z	→	[[matrix$_2$]]
'name' { row col } x	→	

PUTI — Command

Replaces an element in an array or list and increments the position

{ list$_1$ } position obj	→	{ list$_2$ } position'
'name' position obj	→	'name' position'
[vector$_1$] position z	→	[vector$_2$] position'
[[matrix$_1$]] position z	→	[[matrix$_2$]] position'
[[matrix$_1$]] { row col } z	→	[[matrix$_2$]] { row col }'
'name' { row col } x	→	'name' { row col }'

PVARS — Command

Returns list of backup objects and library objects and the type of memory (or amount of memory if independent RAM)

port	→	{ list }	"ROM"
port	→	{ list }	"SYSRAM"
port	→	{ list }	bytes

PVIEW — Command

Displays *PICT* with the specified coordinate or pixel at the upper-left corner. An empty list displays *PICT* centered in the display, ready to scroll.

(x,y)	→	
{ #x #y }	→	
{ }	→	

PWRFIT — Command

Selects power curve-fitting model

PX→C — Command

Pixel to user-unit coordinate conversion

{ #col #row }	→	(x,y)

→Q — Command

Converts numbers to fractional equivalent

x	→	'a/b'
(x,y)	→	'a/b+c/d*i'
'X+1.4'	→	'X+7/5'

Note:

The display mode (*such as 2 FIX*) *affects the result*

→Qπ — Command

→Q after factoring out π

x	→	'a/b*π'
x	→	'a/b'
(x,y)	→	'a/b*π+c/d*π*i'
(x,y)	→	'a/b*π+c/d*i'
(x,y)	→	'a/b+c/d*π*i'
(x,y)	→	'a/b+c/d*i'
'(2.5,3.5)*X'	→	'(5/2+7/2*i)*X'

Note:

The display mode (*such as 2 FIX*) *affects the result*

QUAD Command

Solves a quadratic polynomial

$'symb_1'$ 'global' → $'symb_2'$

QUOTE Command

Returns argument expression unevaluated

'symb' → 'symb'

RAD Command

Sets Radians mode

RAND Command

Returns a random number

→ x

RCEQ Command

Recalls the current equation

→ obj

RCL Command

Recalls the contents of a variable or backup object

'name'	→	obj
PICT	→	grob
:port:name	→	obj
:port:{path name}	→	obj

RCLALARM Command

Recalls alarm from alarm list

n → { date time action repeat }

RCLF Command

Returns a list containing two binary integers representing the system and user flags

→ { #system #user }

Note:

The wordsize should be set to 64 bits

RCLKEYS Command

Lists user-key assignments. S indicates standard keys are active.

→ { obj_1 $rc.p_1$ ... obj_n $rc.p_n$ }

→ { S obj_1 $rc.p_1$ ... obj_n $rc.p_n$ }

RCLMENU — Command

Recalls number and page of active menu

	→	mm.pp

RCWS — Command

Recalls the binary integer wordsize

	→	n

RCLΣ — Command

Recalls the current statistics matrix

	→	obj

RDM — Command

Redimensions a matrix. Extra elements are dropped, missing elements are padded with zeros.

[vector$_1$]	{ cols }	→	[vector$_2$]
[vector]	{ rows cols }	→	[[matrix]]
[[matrix]]	{ cols }	→	[vector]
[[matrix$_1$]]	{ rows cols }	→	[[matrix$_2$]]
'name'	{ cols }	→	
'name'	{ rows cols }	→	

RDZ — Command

Sets the random number seed. Supply 0 to use the system clock.

x	→	

RE — Function

Returns the real part of a complex number, array, or unit object

x	→	x
(x,y)	→	x
[C-array]	→	[R-array]
'symb'	→	'RE(symb)'
x_unit	→	x

RECN — Command

Receives file from remote Kermit, saved in an object named in level 1

'name'	→	
"name"	→	

RECV — Command

Receives file from remote Kermit, saved in a sender-named object

REPEAT — Command

Begins loop clause in WHILE ... REPEAT ... END

T/F	→	

REPL — Command

Replaces the level 1 object onto the level 3 object at the location specified in level 2

{ list } n { sublist }	→	{ list' }
"string" n "substring"	→	"string"
grob (x,y) subgrob	→	grob'
grob { #m #n } subgrob	→	grob'
PICT (x,y) subgrob	→	
PICT { #x #y } subgrob	→	

RES — Command

Sets the plot resolution in user-unit or pixel intervals

n	→	*Interval in user-units*
#n	→	*Interval in pixels*

RESTORE — Command

Replaces HOME directory with backup copy

backup	→	

RL — Command

Rotates left by one bit

$\#n_1$	→	$\#n_2$

RLB — Command

Rotates left by one byte

$\#n_1$	→	$\#n_2$

RND — Function

Rounds fractional part of number

z_1 n	→	z_2
z 'symb'	→	'RND(z,symb)'
'symb' x	→	'RND(symb,x)'
'symb$_1$' 'symb$_2$'	→	'RND(symb$_1$,symb$_2$)'
x_unit n	→	x'_unit
x_unit 'symb'	→	'RND(x_unit,symb)'
[vector$_1$] n	→	[vector$_2$]
[[matrix$_1$]] n	→	[[matrix$_2$]]

RNRM Command

Computes the maximum value of the sums of the absolute values of all elements over all rows

[vector]	$\longrightarrow$	row-norm
[[matrix]]	$\longrightarrow$	row-norm

Note:

Since a vector is considered a 1-row matrix, RNRM returns the largest element in the vector.

ROLL Command

Moves level $n+1$ object to level 1

$\text{obj}_n \ldots \text{obj}_1 \;\; n \longrightarrow \text{obj}_{n-1} \ldots \text{obj}_1 \;\; \text{obj}_n$

ROLLD Command

Moves the level 2 object to level *n*

$\text{obj}_1 \ldots \text{obj}_n \;\; n \longrightarrow \text{obj}_n \;\; \text{obj}_1 \ldots \text{obj}_{n-1}$

ROOT Command

Finds a numerical root

'symb' 'global' guess	$\longrightarrow$	root
'symb' 'global' { guess_1 guess_2 }	$\longrightarrow$	root
'symb' 'global' { guess_1 guess_2 guess_3 }	$\longrightarrow$	root
«program» 'global' guess	$\longrightarrow$	root
«program» 'global' { guess_1 guess_2 }	$\longrightarrow$	root
«program» 'global' { guess_1 guess_2 guess_3 }	$\longrightarrow$	root

ROT Command

Moves the level 3 object to level 1

$\text{obj}_3 \;\; \text{obj}_2 \;\; \text{obj}_1 \longrightarrow \text{obj}_2 \;\; \text{obj}_1 \;\; \text{obj}_3$

RR Command

Rotates right by one bit

$\#n_1 \longrightarrow \#n_2$

RRB Command

Rotates right by one byte

$\#n_1 \longrightarrow \#n_2$

RSD Command

Computes a correction to the solution of a system of equations

[vector **B**] [[matrix **A**]] [vector **Z**] $\rightarrow$ [vector **B**−**AZ**]

[[matrix **B**]] [[matrix **A**]] [[matrix **Z**]] $\rightarrow$ [[matrix **B**−**AZ**]]

R→B Command

Real-to-binary conversion

n $\rightarrow$ #n

R→C Command

Real-to-complex conversion

x y $\rightarrow$ (x,y)

[R-array$_{real}$] [R-array$_{imag}$] $\rightarrow$ [C-array]

R→D Command

Radians-to-degrees conversion

x $\rightarrow$ $(180/\pi)x$

SAME Command

Tests two objects for equality

obj$_1$ obj$_2$ $\rightarrow$ T/F

SBRK Command

Sends serial break

SCALE Command

Specifies x and y scale in units per 10 pixels

x y $\rightarrow$

SCATRPLOT Command

Draws a scatter plot of the data in *ΣDAT*

SCATTER Command

Selects scatter plot

SCI Command

Sets Scientific display mode

n $\rightarrow$

SCONJ Command

Conjugates the contents of a variable

'name' $\rightarrow$

SDEV — Command

Computes standard deviations of the data in ΣDAT

	$\rightarrow$	x
	$\rightarrow$	[x_1 x_2 … x_m]

SEND — Command

Sends object to another Kermit device

'local-name'	$\rightarrow$	
{{ local-name remote-name }}	$\rightarrow$	
{ local-name$_1$ local-name$_2$ … }	$\rightarrow$	
{ { local-name$_1$ remote-name } local-name$_2$ … }	$\rightarrow$	

SERVER — Command

Selects Kermit Server mode

SF — Command

Sets a system or user flag

±n	$\rightarrow$	

SHOW — Command

Resolves all name references or all name references except those in a list

'symb$_1$' 'name'	$\rightarrow$	'symb$_2$'
'symb$_1$' { name }	$\rightarrow$	'symb$_2$'

SIGN — ∫ Function

Sign of a number. Complex numbers return a unit vector in the direction of z.

$x < 0$	$\rightarrow$	-1
$x = 0$	$\rightarrow$	0
$x > 0$	$\rightarrow$	1
z_1	$\rightarrow$	z_2
x_unit	$\rightarrow$	y
'symb'	$\rightarrow$	'SIGN(symb)'

SIN — ↓ ∂ ∫ Function

Sine

z	$\rightarrow$	sin z
'symb'	$\rightarrow$	'SIN(symb)'
x_pa-unit	$\rightarrow$	sin x

SINH

↓∂∫ Function

Hyperbolic sine

z	→	sinh z

SINV

Command

Inverts the contents of a variable

'name'	→	

SIZE

Command

Finds the dimensions of an object

{ list }	→	objects
'algebraic'	→	objects
"string"	→	characters
[vector]	→	{ elements }
[[matrix]]	→	{ rows cols }
grob	→	width height
PICT	→	width height
unit_object	→	objects
other	→	1

SL

Command

Shifts left by one bit

$\#n_1$	→	$\#n_2$

SLB

Command

Shifts left by one byte

$\#n_1$	→	$\#n_2$

SNEG

Command

Negates the contents of a variable

'name'	→	

SQ

↓∂∫ Function

Squares a number or matrix

z	→	z^2
[[matrix]]	→	[[matrix * matrix]]
'symb'	→	'SQ(symb)'
x_unit	→	x^2_unit^2

SR Command

Shifts right by one bit

$\#n_1 \rightarrow \#n_2$

SRB Command

Shifts right by one byte

$\#n_1 \rightarrow \#n_2$

SRECV Command

Reads *n* characters from I/O port. T/F is 1 for successful receive.

n → "string" T/F

START Command

Begins START ... NEXT or START ... STEP

start end **START** loop-clause **NEXT**

start end **START** loop-clause increment **STEP**

STD Command

Sets Standard display mode

STEP Command

Ends FOR ... STEP or START ... STEP

increment →

STEQ Command

Stores into reserved variable *EQ*

obj →

STIME Command

Sets serial transmit/receive timeout. The valid range is 0 to 25.4 seconds. 0 means there is no time limit.

STO Command

Stores an object into a variable

```
          obj  name            →
      obj  :port:name          →
   obj  name(position)         →
           grob  PICT          →
  backup  port-number          →
 library  port-number          →
```

STOALARM Command

Stores alarm in system alarm list

time	→	alarm-number
{ date }	→	alarm-number
{ date time }	→	alarm-number
{ date time action }	→	alarm-number
{ date time action repeat }	→	alarm-number

STOF Command

Sets the system flags or the system and user flags according to the value of two binary integers in a list

#system	→	
{ #system #user }	→	

Note:

The wordsize should be set to 64 bits

STOKEYS Command

Makes multiple user-key assignments. Including S activates standard key definitions.

S	→	
{ obj_1 rc.p_1 ... obj_n rc.p_n }	→	
{ S obj_1 rc.p_1 ... obj_n rc.p_n }	→	

STO+ Command

Storage addition (see +)

object 'name'	→	
'name' object	→	

STO- Command

Storage subtraction (see -)

object 'name'	→	
'name' object	→	

STO* Command

Storage multiplication (see *)

object 'name'	→	
'name' object	→	

STO/ Command

Storage division (see /)

object 'name'	→	
'name' object	→	

STOΣ Command

Stores into reserved variable *ΣDAT*

obj	→	

STR→ Command

Evaluates the commands defined by a string after removing the " " delimiters

"string"	→	

→STR Command

Converts an object to a string

object	→	"object"

STWS Command

Sets the binary integer wordsize

n	→	
#n	→	

SUB Command

Extracts a portion of a list, string, or grob

{ list } start end	→	{ sublist }
"string" start end	→	"substring"
grob (x_1,y_1) (x_2,y_2)	→	subgrob
grob { #x_1 #y_1 } { #x_2 #y_2 }	→	subgrob
PICT (x_1,y_1) (x_2,y_2)	→	subgrob
PICT { #x_1 #y_1 } { #x_2 #y_2 }	→	subgrob

SWAP Command

Swaps the objects in levels 1 and 2

obj_2 obj_1	→	obj_1 obj_2

SYSEVAL Command

Executes a system object

#n	→	

→TAG Command

Tags an object with another object

obj "tag"	→	:tag:obj
obj 'name'	→	:name:obj
obj x	→	:x:obj

TAN ↓ ∂ ∫ Function

Tangent

z	→	tan z
'symb'	→	'TAN(symb)'
x_pa-unit	→	tan x

TANH ↓ ∂ ∫ Function

Hyperbolic tangent

z	→	tanh z

TAYLR Command

Computes a Taylor series approximation

'symb$_1$' 'global' degree	→	'symb$_2$'

TEXT Command

Selects the stack display

THEN Command

Begins true-clause of IF, IFERR, or CASE structures

T/F	→	

TICKS Command

Returns time in binary integer clock ticks (8192 per second)

	→	#n

TIME Command

Returns current time as number

	→	HH.MMSS

→TIME Command

Sets specified system time

HH.MMSS	→	

TLINE Command

Toggles pixels on a straight line

(x,y) (x',y')	→	
{ #x$_1$ #y$_1$ } { #x$_2$ #y$_2$ }	→	

TMENU — Command

Displays temporary built-in or list-defined menu (see *Menus*)

mm.pp	$\rightarrow$	
'list-name'	$\rightarrow$	
{ *names and commands* }	$\rightarrow$	

Note:

TMENU does not affect the contents of the variable CST

TOT — Command

Sums the columns in *ΣDAT*

	$\rightarrow$	x
	$\rightarrow$	[x_1 x_2 ... x_m]

TRANSIO — Command

Selects character translation mode

n	$\rightarrow$	
0		*No translation*
1		*CR to CR/LF (default)*
2		*Chars 128-159*
3		*Chars 128-255*

TRN — Command

Transposes a matrix

[[matrix_1]]	$\rightarrow$	[[matrix_2]]
'name'	$\rightarrow$	

TRNC — Command

Truncates number

z_1	n	$\rightarrow$	z_2
[vector_1]	n	$\rightarrow$	[vector_2]
[[matrix_1]]	n	$\rightarrow$	[[matrix_2]]
x_1_unit	n	$\rightarrow$	x_2_unit

TRUTH — Command

Selects truth plot

TSTR — Command

Converts date and time numbers to string form

date	time	$\rightarrow$	"string"

TVARS	Command

Lists the variables of specified type found in the current directory (see *Object Types*)

type	→	{ names }
{ type_1 type_2 ... }	→	{ names }

TYPE	Command

Returns the type of an object (see *Object Types*)

object	→	type

UBASE	Function

Converts unit object to SI base units

x	→	x
'symb'	→	'UBASE(symb)'
x_units	→	y_base-units

UFACT	Command

Factors specified compound unit

x y_units	→	x
x_units_1 y_units_2	→	$\text{x}'\text{\_units}_2$ * units_3

→UNIT	Command

Combines number and unit object to create a new unit object

x y_units	→	x_units

UNTIL	Command

Begins test-clause of DO ... UNTIL ... END

UPDIR	Command

Makes parent directory the current directory

UTPC	Command

Upper-tail Chi-Square distribution

d.o.f. x	→	utpc(d,x)

UTPF	Command

Upper-tail F-distribution

d.o.f._1 d.o.f._2 x	→	utpf(d.o.f._1, d.o.f._2, x)

UTPN	Command

Upper-tail normal distribution

mean variance x	→	utpn(mean, variance, x)

UTPT Command

Upper-tail t-distribution

d.o.f. x	→	utpt(d.o.f.,x)

UVAL Function

Returns scalar portion of unit object

x	→	x
'symb'	→	'UVAL(symb)'
x_unit	→	x

→V2 Command

Combines two real numbers into 2-D vector or complex number according to flag -19 and the current Coordinate System (flags -15 and -16)

x y	→	[x y]
x y	→	[x ∡y]
x y	→	(x,y)
x y	→	(x,∡y)

→V3 Command

Combines three real numbers into 3-D vector according to the current Coordinate System (flags -15 and -16)

x y z	→	[x y z]
x y_θ z	→	[x ∡y_θ z]
x y_θ z_ϕ	→	[x ∡y_θ ∡z_ϕ]

VAR Command

Variances of Σ*DAT* data in columns specified by COLΣ

	→	x
	→	[x_1 x_2 ... x_m]

VARS Command

Returns list of variables in the current directory

	→	{ names }

VTYPE Command

Returns the type of an object in the named variable, or -1 if the variable is nonexistent (see *Object Types*)

'name'	→	type
:port:name	→	type

V→ Command

Separates a 2 or 3 element vector. If there are more than 3 elements, the current Coordinate System (flags −15 and −16) is ignored.

$[\,x\ y\,]$	→	$x\quad y$
$[\,x_r\ \measuredangle y_\theta\,]$	→	$x_r\quad y_\theta$
$[\,x\ y\ z\,]$	→	$x\quad y\quad z$
$[\,x_r\ \measuredangle y_\theta\ z\,]$	→	$x_r\quad y_\theta\quad z$
$[\,x_r\ \measuredangle y_\theta\ \measuredangle z_\phi\,]$	→	$x_r\quad y_\theta\quad z_\phi$
$(\,x,y\,)$	→	$x\quad y$
$(\,x_r,\measuredangle y\,)$	→	$x_r\quad \measuredangle y_\theta$
$[\,x_1\ x_2 \ldots x_n\,]$	→	$x_1\ x_2 \ldots x_n$

WAIT Command

Pauses program execution or waits for a key

seconds	→	
0	→	rc.p *Doesn't update menu*
−1	→	rc.p *Displays current menu*

WHILE Command

Begins WHILE ... REPEAT ... END

WHILE test−clause **REPEAT** loop−clause **END**

WSLOG Command

Returns four strings indicating the time, date, and source of the four most recent system halts (see *System Operations*)

→ "string_4" "string_3" "string_2" "string_1"

XCOL Command

Specifies *ΣDAT* column as the independent variable

x−column →

XMIT Command

Sends string through I/O port without Kermit

"string"	→	1
"string"	→	"unsent string" 0

XOR

Function

Logical or binary XOR

#n_1 #n_2	→	#n_3
x y	→	T/F
x 'symb'	→	'x XOR symb'
'symb' x	→	'symb XOR x'
'$symb_1$' '$symb_2$'	→	'$symb_1$ XOR $symb_2$'
"$string_1$" "$string_2$"	→	"$string_3$"

Note:

String arguments must have the same length

XPON

Function

Returns the exponent of a number

x	→	n
'symb'	→	'XPON(symb)'

XRNG

Command

Specifies x-axis plotting range

x_{min} x_{max}	→	

XROOT

Function

Returns x^th root of y

y x	→	$\sqrt[x]{y}$
y x_pa-unit	→	y'
y_unit x	→	$\sqrt[x]{y}$_unit$^{\frac{1}{x}}$
y 'symb'	→	'XROOT(symb,y)'
'symb' x	→	'XROOT(x,symb)'
'$symb_1$' '$symb_2$'	→	'XROOT($symb_2$,$symb_1$)'
y_pa-unit x_unit	→	y'_unit'
'$symb_1$' x_pa-unit	→	'XROOT(x_pa-unit,$symb_1$)'
y_unit 'symb'	→	'XROOT(symb,y_unit)'

YCOL

Command

Specifies a *ΣDAT* column as the dependent variable

y-column	→	

YRNG

Command

Specifies y-axis plotting range

y_{min} y_{max}	→	

***H** Command

Adjusts the height of a plot. Enlarges (zooms out) if factor > 1.

factor →

***W** Command

Adjusts the width of a plot. Enlarges (zooms out) if factor > 1.

factor →

√ ↓ ∂ ∫ Function

Square root

z	→	sqrt z
'symb'	→	'√(symb)'
x_unit	→	$x^{.5}$_$unit^{.5}$

∫ ∂ ∫ Function

Integral

lower-limit upper-limit 'integrand' 'name' → integral

'∫(lower-limit, upper-limit, integrand, name)'

Notes:

1) name *is the variable of integration.*
2) *Set Numerical Results mode (flag −3) to perform a numerical integration on the stack.*
3) *The display mode (such as 2 FIX) specifies the accuracy factor for numerical integration, and the uncertainty of integration is stored in reserved variable IERR.*

∂ ∂ ∫ Function

Derivative

'$symb_1$' 'name' → '$symb_2$' *Complete*

'∂*name*(expression)' *Stepwise*

Note:

name *is the variable of differentiation.*

π ∂ Function

Symbolic constant π

→ 'π'

Σ ∂ Function

Summation

'summation-index' initial-value final-value 'summand' → sum

'Σ(summation-index=initial-value,final-value, summand)'

ΣLINE Command

Returns best-fit line for data in *ΣDAT* with values for *a* and *b* filled in

Linear model	→	'a+b*X'
Logarithmic model	→	'a+b*LN(X)'
Exponential model	→	'a*EXP(b*X)'
Power model	→	'a*X^b'

ΣX Command

Sum of data of data in independent *ΣDAT* column

→ ΣX_i

ΣX^2 Command

Sum of squares of data in independent *ΣDAT* column

→ ΣX_i^2

ΣY Command

Sum of data in dependent *ΣDAT* column

→ ΣY_i

ΣY^2 Command

Sum of squares of data in dependent *ΣDAT* column

→ ΣY_i^2

ΣX*Y Command

Sum of products of data in independent and dependent *ΣDAT* columns

→ $\Sigma X_i Y_i$

Σ+ Command

Appends one or more data points to *ΣDAT*

x	→	
[vector]	→	
[[matrix]]	→	

Σ- Command

Deletes last row from *ΣDAT*

	→	x
	→	[vector]

< Function

Less–than comparison

x y	→	x<y (T/F)
x y_pa-unit	→	T/F
x_pa-unit y	→	T/F
x_unit_1 y_unit_2	→	T/F
x 'symb'	→	'x<symb'
'symb' x	→	'symb<x'
'symb' x_unit	→	'symb<x_unit'
x_unit 'symb'	→	'x_unit>symb'
'$symb_1$' '$symb_2$'	→	'$symb_1<symb_2$'
:tag:object object	→	T/F
object :tag:object	→	T/F
object object	→	T/F

Notes:

1) Units must be dimensionally consistent

2) Tags are dropped before the comparison

> Function

Greater–than comparison

x y	→	x>y (T/F)
x y_pa-unit	→	T/F
x_pa-unit y	→	T/F
x_unit_1 y_unit_2	→	T/F
x 'symb'	→	'x>symb'
'symb' x	→	'symb>x'
x_unit 'symb'	→	'x_unit>symb'
'$symb_1$' '$symb_2$'	→	'$symb_1>symb_2$'
'symb' x_unit	→	'symb>x_unit'
:tag:object object	→	T/F
object :tag:object	→	T/F
object object	→	T/F

Notes:

1) Units must be dimensionally consistent

2) Tags are dropped before the comparison

$\leq$ Function

Less-than-or-equal comparison

x y	$\rightarrow$	$x \leq y$ (T/F)
x y_pa-unit	$\rightarrow$	T/F
x_pa-unit y	$\rightarrow$	T/F
x_unit_1 y_unit_2	$\rightarrow$	T/F
x 'symb'	$\rightarrow$	'x $\leq$ symb'
'symb' x	$\rightarrow$	'symb $\leq$ x'
'symb' x_unit	$\rightarrow$	'symb $\leq$ x_unit'
x_unit 'symb'	$\rightarrow$	'x_unit $\leq$ symb'
'$symb_1$' '$symb_2$'	$\rightarrow$	'$symb_1 \leq symb_2$'
:tag:object object	$\rightarrow$	T/F
object :tag:object	$\rightarrow$	T/F
object object	$\rightarrow$	T/F

Notes:

1) Units must be dimensionally consistent

2) Tags are dropped before the comparison

$\geq$ Function

Greater-than-or-equal comparison

x y	$\rightarrow$	$x \geq y$ (T/F)
x y_pa-unit	$\rightarrow$	T/F
x_pa-unit y	$\rightarrow$	T/F
x_unit_1 y_unit_2	$\rightarrow$	T/F
x 'symb'	$\rightarrow$	'x $\geq$ symb'
'symb' x	$\rightarrow$	'symb $\geq$ x'
'symb' x_unit	$\rightarrow$	'symb $\geq$ x_unit'
x_unit 'symb'	$\rightarrow$	'x_unit $\geq$ symb'
'$symb_1$' '$symb_2$'	$\rightarrow$	'$symb_1 \geq symb_2$'
:tag:object object	$\rightarrow$	T/F
object :tag:object	$\rightarrow$	T/F
object object	$\rightarrow$	T/F

Notes:

1) Units must be dimensionally consistent

2) Tags are dropped before the comparison

$\neq$ Function

Not – equal comparison

x	y	→	$x \neq y$ (T/F)
x	z	→	T/F
z	x	→	T/F
x	y_pa-unit	→	T/F
x_pa-unit	y	→	T/F
x_unit$_1$	y_unit$_2$	→	T/F
z	'symb'	→	'z $\neq$ symb'
'symb'	z	→	'symb $\neq$ z'
'symb'	x_unit	→	'symb $\neq$ x_unit'
x_unit	'symb'	→	'x_unit $\neq$ symb'
'symb$_1$'	'symb$_2$'	→	'symb$_1$ $\neq$ symb$_2$'
:tag:object	object	→	T/F
object	:tag:object	→	T/F
object	object	→	T/F

Notes:

1) Units must be dimensionally consistent

2) Real – complex comparisons assume the imaginary part is 0

3) Tags are dropped before the comparison

== Function

Logical equality comparison

x y	$\rightarrow$	x==y (T/F)
x z	$\rightarrow$	T/F
z x	$\rightarrow$	T/F
x y_pa-unit	$\rightarrow$	T/F
x_pa-unit y	$\rightarrow$	T/F
x_unit_1 y_unit_2	$\rightarrow$	T/F
z 'symb'	$\rightarrow$	'z==symb'
'symb' z	$\rightarrow$	'symb==z'
'symb' x_unit	$\rightarrow$	T/F
x_unit 'symb'	$\rightarrow$	'x_unit == symb'
'symb' x_unit	$\rightarrow$	'symb == x_unit'
'$symb_1$' '$symb_2$'	$\rightarrow$	'$symb_1$==$symb_2$'
:tag:object object	$\rightarrow$	T/F
object :tag:object	$\rightarrow$	T/F
object object	$\rightarrow$	T/F

Notes:

1) *Units must be dimensionally consistent*

2) *Real – complex comparisons assume the imaginary part is 0*

3) *Tags are dropped before the comparison*

→ Command

Assigns local variable(s)

$obj_1 \ldots obj_n \rightarrow$

+ ↓∂ Function

Adds two objects

z_1 z_2	→	$z_1 + z_2$
#n m	→	#n+m
n #m	→	#n+m
#n #m	→	#n+m
x_unit y_unit	→	x+y_unit
x y_pa-unit	→	x+y_pa-unit
x_pa-unit y	→	x+y
'$symb_1$' '$symb_2$'	→	'$symb_1$ + $symb_2$'
z 'symb'	→	'z+symb'
'symb' z	→	'symb+z'
'symb' x_unit	→	'symb+x_unit'
x_unit 'symb'	→	'x_unit+symb'
[$vector_1$] [$vector_2$]	→	[$vector_1$ + $vector_2$]
[[$matrix_1$]] [[$matrix_2$]]	→	[[$matrix_1$ + $matrix_2$]]
$grob_1$ $grob_2$	→	$grob_3$
{$list_1$} {$list_2$}	→	{$list_1$ $list_2$}
"abc" "def"	→	"abcdef"
{ list } object	→	{ list object }
object { list }	→	{ object list }
"string" object	→	"stringobject"
object "string"	→	"objectstring"

Notes:

1) Grobs must have identical dimensions.

2) →STR is executed on objects added to strings.

3) Units must be dimensionally consistent

– ↓∂ Function

Subtracts two objects

z_1 z_2	→	$z_1 - z_2$
#n m	→	#n–m
n #m	→	#n–m
#n #m	→	#n–m
x_unit y_unit	→	x–y_unit
x y_pa-unit	→	x–y_pa-unit
x_pa-unit y	→	x–y
z 'symb'	→	'z–symb'
'symb' z	→	'symb–z'
'symb$_1$' 'symb$_2$'	→	'symb$_1$ – symb$_2$'
'symb' x_unit	→	'symb–x_unit'
x_unit 'symb'	→	'x_unit–symb'
[vector$_1$] [vector$_2$]	→	[vector$_1$–vector$_2$]
[[matrix$_1$]] [[matrix$_2$]]	→	[[matrix$_1$–matrix$_2$]]

Note:

Units must be dimensionally consistent

* ↓∂ Function

Multiplies two objects

z_1 z_2	→	z_1*z_2
#n #m	→	#n*m
#n m	→	#n*m
n #m	→	#n*m
[vector] z	→	[vector*z]
z [vector]	→	[vector*z]
[[matrix]] [vector]	→	[matrix*vector]
[[matrix]] [[matrix]]	→	[[matrix*matrix]]
z 'symb'	→	'z*symb'
'symb' z	→	'symb*z'
'symb$_1$' 'symb$_2$'	→	'(symb$_1$)*(symb$_2$)'
x_unit$_1$ y_unit$_2$	→	x*y_unit$_3$
x y_unit	→	x*y_unit
x_unit y	→	x*y_unit
x_unit 'symb'	→	'(x_unit)*(symb)'
'symb' x_unit	→	'(symb)*(x_unit)'

/

↓ ∂ Function

Divides two objects

z_1 z_2	→	z_1 / z_2
n #m	→	#n/m
#n m	→	#n/m
#n #m	→	#n/m
[vector] z	→	[vector/z]
[vector] [[matrix]]	→	[[vector/matrix]]
z 'symb'	→	'z/(symb)'
'symb' z	→	'(symb)/z'
'symb$_1$' 'symb$_2$'	→	'(symb$_1$)/(symb$_2$)'
x_unit$_1$ y_unit$_2$	→	x/y_unit$_1$/unit$_2$
x y_unit	→	x/y_1/unit
x_unit y	→	x/y_unit
x_unit 'symb'	→	'(x_unit)/(symb)'
'symb' x_unit	→	'(symb)/(x_unit)'

^

↓ ∂ ∫ Function

Raises a number to a power

z_1 z_2	→	z_1 ^ z_2
z 'symb'	→	'z^(symb)'
'symb' z	→	'(symb)^z'
'symb$_1$' 'symb$_2$'	→	'(symb$_1$)^(symb$_2$)'
x_unit y_pa-unit	→	x^y_unit
x y_pa-unit	→	x'
x_unit y	→	x^y_unit^y
x_unit 'symb'	→	'(x_unit)^(symb)'
'symb' x_unit	→	'(symb)^(x_unit)'

!

Function

Factorial or gamma function

n	→	n!
x	→	Γ(x+1)
'symb'	→	'(symb)!'

| (where)

∂ Function

Substitutes symbolics for names in a symbolic expression

'symb$_{old}$' { name$_1$ symb$_1$ ... name$_n$ symb$_n$ } → 'symb$_{new}$'

z { name$_1$ symb$_1$ ... name$_n$ symb$_n$ } → z

'symb$_{old}$ | (name$_1$=symb$_1$, ..., name$_n$=symb$_n$)'

% Function

Percent

x y	→	xy/100
x 'symb'	→	'%(x,symb)'
'symb' x	→	'%(symb,x)'
'symb$_1$' 'symb$_2$'	→	'%(symb$_1$,symb$_2$)'
x_unit y	→	xy/100_unit
x_unit 'symb'	→	'%(x_unit,symb)'
'symb' x_unit	→	'%(symb,x_unit)'
x y_unit	→	xy/100_unit

%CH Function

Percent change

x y	→	100(y−x)/x
x 'symb'	→	'%CH(x,symb)'
'symb' x	→	'%CH(symb,x)'
'symb$_1$' 'symb$_2$'	→	'%CH(symb$_1$,symb$_2$)'
x_unit y_unit	→	100(y−x)/x
x y_pa-unit	→	100(y'−x)/x
x_pa-unit y	→	100(y−x')/x'
x_unit 'symb'	→	'%CH(x_unit,symb)'
'symb' x_unit	→	'%CH(symb,x_unit)'

Note:

Units must be dimensionally consistent

%T Function

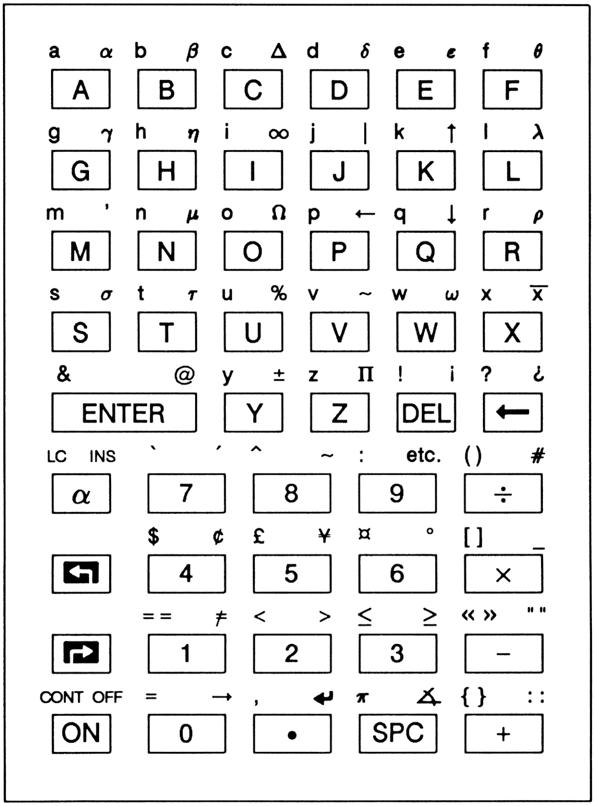
Percent total

x y	→	100y/x
x 'symb'	→	'%T(x,symb)'
'symb' x	→	'%T(symb,x)'
'symb$_1$' 'symb$_2$'	→	'%T(symb$_1$,symb$_2$)'
x_unit y_unit	→	100y/x
x y_pa-unit	→	100y'/x
x_pa-unit y	→	100y/x'
x_unit 'symb'	→	'%T(x_unit,symb)'
'symb' x_unit	→	'%T(symb,x_unit)'

Note:

Units must be dimensionally consistent

Alpha Keyboard



The HP 48 Handbook

ISBN 1-879828-00-6