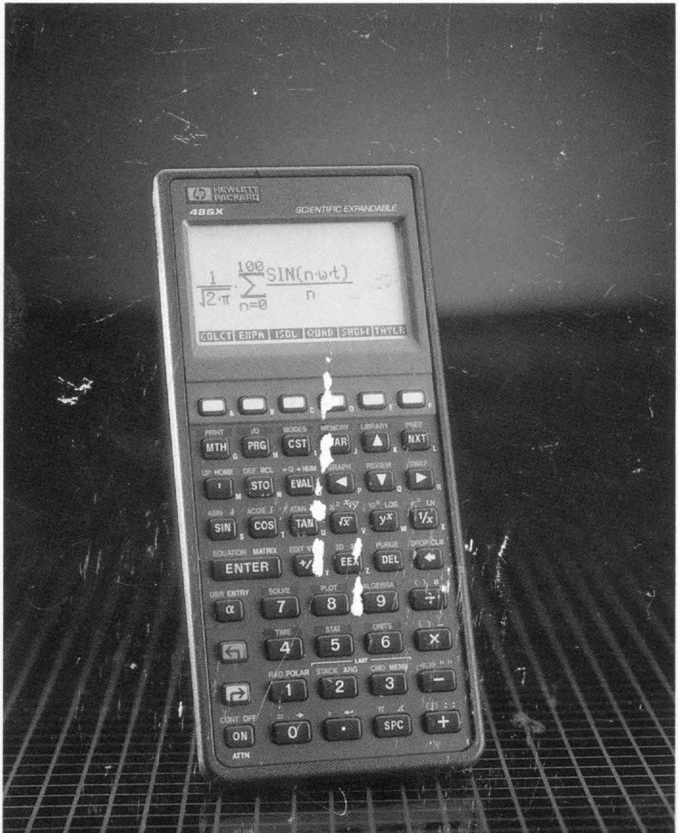
The HP 48 Programmer's ToolKit



James Donnelly

JOSEPH K. HORN

The HP 48

Programmer's ToolKit

James Donnelly

The software, this manual, and any examples contained herein are provided "as is" and are subject to change without notice. James Donnelly makes no warranty of any kind with regard to this software or manual, including, but not limited to, the implied warranties of merchantability and fitness for any purpose. James Donnelly shall not be liable for any error or for incidental or consequential damages in connection with the furnishing, performance, or use of this software, manual, or examples presented herein.

First Edition

First Printing, July 1990

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

Acknowledgements

Special thanks go to Dan Allen, Scott Burke, Ray Depew, Alonzo Gariepy, Wlodek A.C. Mier-Jedrzejowicz, Bob Moore, Richard Nelson, Jake Schwartz, Eric L. Vogel, Frank Wales, and Dennis York for their advice, ideas, support, and encouragement.

The cover photograph was provided by the Hewlett-Packard Company.

Contents

Getting Started

The HP 48 Programmer's ToolKit is a collection of software tools designed with the programmer in mind. These tools improve program performance by combining some common, slow operations into faster internal system languages and provide additional capability in object manipulation not directly available in the HP 48 command set.

There are seven main chapters and several reference tables in this manual:

- *Character Set Catalog* describes an interactive character set catalog.
- *Menu Label Builder* describes an interactive program for building graphics objects for use in custom menus.
- *Flag Catalog* describes the interactive Flag Catalog.
- *Data Browser* and *Title Browser* describe two powerful screen-oriented user interface utilities that may be used to enhance the appearance of an application.
- *Tool Library* describes the new commands provided in the Tool Library, including the meta-object concepts used by some of the new commands.
- *Command Reference* describes the full syntax for each new command in the Tool Library with examples.
- Additional chapters provide reference tables for object types, the character set, and flags.

Additional Information

Part 5 of *The HP 48 Owner's Manual* discusses data transfer. The documentation that comes with the Serial Interface Kit for an IBM-compatible personal computer (HP 82208A) or an Apple Macintosh computer (HP 82209A) may also be helpful.

The HP 48 Handbook by the same author contains complete stack diagrams for all the HP 48 commands, further discussions of graphics, menus, data transfer topics, and includes other helpful information and reference tables.

Installing the ToolKit

The HP 48 Programmer's ToolKit consists of two system programs and four library objects that extend the built-in command set. All objects must be downloaded in binary mode.

Name	Type	Lib Id	Description
CSCAT	System Program		Character Set Catalog
LBLD	System Program		Menu Label Builder
FCLIB	Library	775	Flag Catalog
TLLIB	Library	776	Tool Library
TBLIB	Library	777	Title Browser
DBLIB	Library	778	Data Browser

Installing System Programs

The system programs CSCAT and LBLD are implemented in system languages and must be downloaded to the HP 48 in binary mode. They may be stored in any variable and evaluated like any other program.

Note: When CSCAT and LBLD are on the stack, the copyright message appears. The programs may not be edited. If you press [EDIT] or [▼] while they are in level one, the HP 48 will take a very long time to display the programs in the command line. Pressing [ENTER] thereafter will only result in a Syntax Error. To abort the long display delay, just press [ATTN].

Installing Library Objects

Libraries are referenced by a *library#* or a library identifier (**:** *port#* **:** *library#*), depending on the command. The title of a library may be displayed by pressing ↰ REVIEW in the LIBRARY menu.

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 install a ToolKit library, perform the following:

- Download the library to the HP 48 in binary mode.
- Recall the library to the stack.
- Purge the variable that the library was stored in.
- Store the library object in a port, such as port 0. For instance, when 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. All ToolKit libraries automatically attach themselves to the HOME directory.

Removing the ToolKit

To remove the Alpha Catalog and the Label Builder, just purge the variable in which they are stored. To remove ToolKit libraries, perform the following steps:

- Ensure that the library object to be purged does not appear on the stack as `Library nnn:` ... Either store the library in a variable or execute NEWOB to create a unique copy.
- The ToolKit libraries are attached to the HOME directory. Switch to the HOME directory, enter the port-tagged library number, such as `:0:775` and execute DETACH.
- Enter the library number as above and execute PURGE.

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.

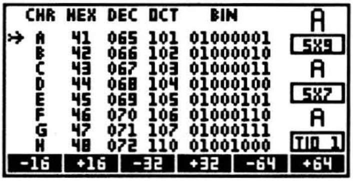
All characters in the programs are case-sensitive. The names of commands are always uppercase. By convention, the names of global variables are uppercase and of local variables are lowercase.

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 type a program into the HP 48, use the BYTES command 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. Named programs may be found on the disk.

Character Set Catalog

The Character Set Catalog provides an interactive character set catalog (see *Character Codes*). To display the Character Set Catalog, execute the system program CSCAT:

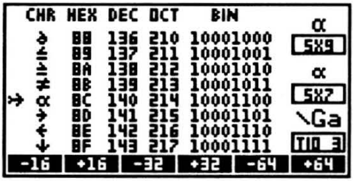


```
   CHR HEX DEC OCT    BIN        A
→  A   41  065 101 01000001   5X9
   B   42  066 102 01000010
   C   43  067 103 01000011     A
   D   44  068 104 01000100   5X7
   E   45  069 105 01000101
   F   46  070 106 01000110     A
   G   47  071 107 01000111
   H   48  072 110 01001000   TIO 1
 -16 | +16 | -32 | +32 | -64 | +64
```

The display above shows eight characters at a time. Each character is shown with its character code displayed in HEX, DECimal, OCTal, and BINary modes. The right side of the display shows the character in three additional forms:

- The character in the large (5x9) font.
- The character in the medium (5x7) font.
- Translated using the current TRANSIO setting. The TIO 1 label reflects the current TRANSIO setting.

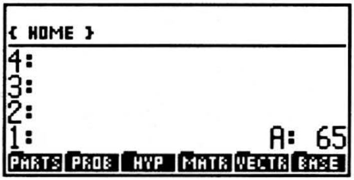
The display above assumes the current TRANSIO setting is 1. The display below shows the display with character codes 136–143 displayed and TRANSIO set to 3:



```
   CHR HEX DEC OCT    BIN        α
   ∂   88  136 210 10001000   5X9
   ≤   89  137 211 10001001
   ≥   8A  138 212 10001010     α
   ≠   8B  139 213 10001011   5X7
→  α   8C  140 214 10001100
   →   8D  141 215 10001101   \Ga
   ←   8E  142 216 10001110
   ↓   8F  143 217 10001111   TIO 3
 -16 | +16 | -32 | +32 | -64 | +64
```

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

- Press the arrow keys to move the pointer. The left shifted arrow keys move a screen (8 characters) at a time. The right shifted arrow keys move to character codes 0 or 255.
- Press the -16 or +16 menu keys to move the pointer backwards or forwards 16 characters.
- Press the -32 or +32 menu keys to move the pointer backwards or forwards 32 characters. For A→Z, +32 yields lowercase a→z.
- Press the -64 or +64 menu keys to move the pointer backwards or forwards 64 characters. For A→Z, -64 yields control codes control-A→control-Z.
- Press ENTER to return the character code to the stack as an alpha-tagged character code, such as A: 65:



```
{ HOME }
4:
3:
2:
1:                    A: 65
```

By executing OBJ→ on the result, the character and its code are available as separate objects.

- Press ATTN to end the application.

Note: The current TRANSIO setting is stored in the reserved variable *IOPAR*. If this variable does not exist, CSCAT will create *IOPAR* in the HOME directory with default values.

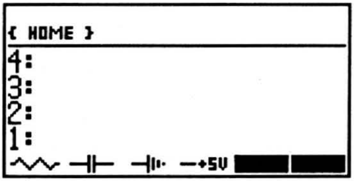
Menu Label Builder

The Label Builder has been designed to facilitate the creation of graphic menu key labels. It is used in conjunction with custom menu definitions supplied to the MENU or TMENU commands which may contain a 21x8 graphics object to define the menu label.

Example: The following list contains a menu definition for four keys. Each key is labeled with a graphics object, and the first key has a different definition for the left and right shifts:

ELEC (225 bytes, checksum #9447h)

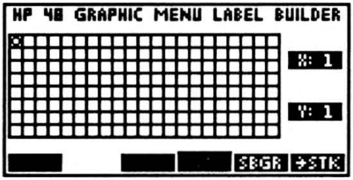
```
{
 {
   GROB 21 8 000000000000101008282006444C00828200010100000000
   { "10Ω" "200Ω" "500Ω" }
 }
 {
   GROB 21 8 00000000A00000A00000A000CFBF7000A00000A00000A000
   "CAPACITOR"
 }
 {
   GROB 21 8 000000004000004100004500 0F751000450000410000 4000
   "GROUND"
 }
 {
   GROB 21 8 000000000000000750002150E77750002450000720000000
   "VCC"
 }
}
```



```
{ HOME }
4:
3:
2:
1:
                   +5V
```

The Label Builder. The variable LBLD contains the Label Builder. While primarily intended for creating graphic menu labels, the Label Builder is also useful for creating smaller graphics objects as well.

To start the Label Builder, execute LBLD:

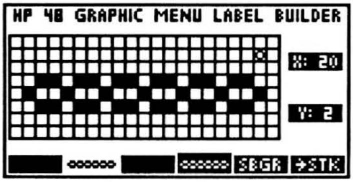


The cursor appears in the upper-left corner of the grid, and the cursor coordinates are shown on the right side of the display.

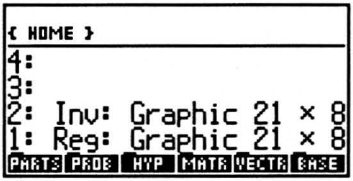
While the grid is displayed, you can do the following:

- Press the arrow keys to move the cursor (wraparound is enabled).
- Press ENTER to toggle the current state of a pixel.
- Press SBGR to return the subgrob defined by the upper-left corner and the cursor to the stack.
- Press →STK to return the menu key graphics object and its inverse to the stack.
- Press ATTN to end the label builder.

The second and fourth menu keys at the bottom of the display show how the menu key would appear in its final form:



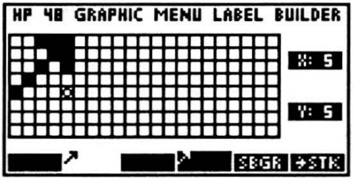
The Label Builder returns a graphics object and its inverse to the stack:



```
{ HOME }
4:
3:
2: Inv: Graphic 21 × 8
1: Reg: Graphic 21 × 8
```

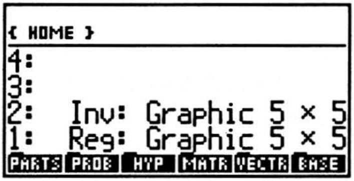
These graphics objects are ready to supply to a custom menu definition. The object returned to level 1 with the tag "Reg" represents the second menu key from the left in the builder; the level 2 object represents the fourth menu key.

The Label Builder may also be used to prepare smaller graphics objects. For instance, to construct a small arrow, set the desired pixels and place the cursor on the lower – right pixel:



```
HP 48 GRAPHIC MENU LABEL BUILDER
X: 5
Y: 5
```

Press SBGR to return the smaller graphics object to the stack, then [ATTN] to end the application. The graphics objects in levels one and two contain the arrow:



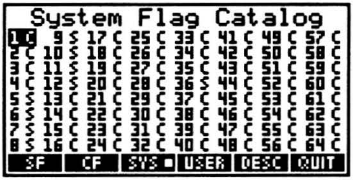
```
{ HOME }
4:
3:
2:  Inv: Graphic 5 × 5
1:  Reg: Graphic 5 × 5
```

Flag Catalog

The Flag Catalog provides a rapid view of all the system flags (−1 − −64) and the user flags (1 − 64). To display the Flag Catalog, execute the command FCAT ([⬅] [LIBRARY] FCLIB FCAT).

Viewing All Flag Settings

When the Flag Catalog starts, the first display shows all the system flags:



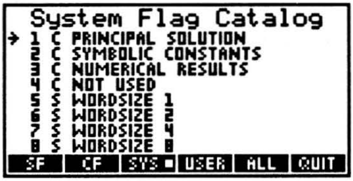
```
  System Flag Catalog
 1 C  9 S 17 C 25 C 33 C 41 C 49 C 57 C
 2 C 10 S 18 C 26 C 34 C 42 C 50 C 58 C
 3 C 11 S 19 C 27 C 35 C 43 C 51 C 59 C
 4 C 12 S 20 C 28 C 36 S 44 C 52 C 60 C
 5 S 13 C 21 C 29 C 37 C 45 C 53 C 61 C
 6 S 14 C 22 C 30 C 38 C 46 C 54 C 62 C
 7 S 15 C 23 C 31 C 39 C 47 C 55 C 63 C
 8 S 16 C 24 C 32 C 40 C 48 C 56 C 64 C
```

This display show all the system flags or user flags at once. When all the flags are displayed, you can do the following:

- Press the arrow keys to move around the display. The left-shifted arrow keys move to the boundaries, and the right-shifted arrow keys move to the first or last flags.
- Press the SF or CF menu keys to set or clear the indicated flag.
- Press the SYS or USER menu keys to view either the system flags or user flags.
- Press the DESC menu key to display the flag descriptions.
- Press QUIT or [ATTN] to end the application.

Viewing Flag Descriptions

The DESC key displays the flag descriptions for either the system flags or user flags:



```
 System Flag Catalog
→ 1 C PRINCIPAL SOLUTION
  2 C SYMBOLIC CONSTANTS
  3 C NUMERICAL RESULTS
  4 C NOT USED
  5 S WORDSIZE 1
  6 S WORDSIZE 2
  7 S WORDSIZE 4
  8 S WORDSIZE 8
```

When the flag descriptions are displayed, you can do the following:

- Press the arrow keys to move the pointer. The left-shifted arrow keys move a screen at a time. The right-shifted arrow keys move to the ends of the list.
- Press the SF or CF menu keys to set or clear the indicated flag.
- Press the SYS or USER menu keys to view either the system flags or user flags.
- Press the ALL menu key to display the flag descriptions.
- Press [ENTER] to toggle the state of a flag. (no!)
- Press QUIT or [ATTN] to end the application.

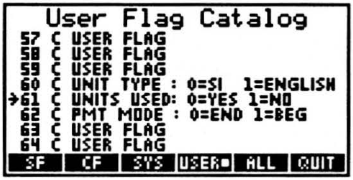
Supplying User Flag Descriptions

When the user flag descriptions are displayed, the current path is searched for the variable *UFLAGS*. If *UFLAGS* is a list containing strings, the first two characters of each string will be examined for a flag number, and the remainder of the string will be displayed as the flag description.

The following list supplies definitions for the flags used by the HP 82211A Solve Equation Library application card:

UFLAGS (107.5 bytes, checksum #7BABh)

```
        {
         "60UNIT TYPE : 0=SI  1=ENGLISH"
         "61UNITS USED: 0=YES 1=NO"
         "62PMT MODE : 0=END 1=BEG"
        }
```



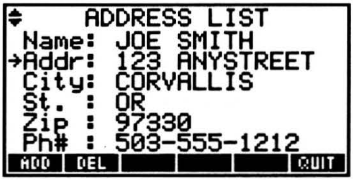
```
    User Flag Catalog
 57 C USER FLAG
 58 C USER FLAG
 59 C USER FLAG
 60 C UNIT TYPE : 0=SI  1=ENGLISH
→61 C UNITS USED: 0=YES 1=NO
 62 C PMT MODE : 0=END 1=BEG
 63 C USER FLAG
 64 C USER FLAG
```

If *UFLAGS* does not contain a list, or the list does not contain a valid string definition, *UFLAGS* will be ignored.

Data Browser

The Data Browser is a utility which provides an efficient interface for examining and editing a series of objects.

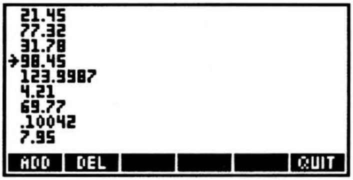
The Data Browser appears to the user as a list of optionally-labeled data with a movable pointer to indicate a choice:



```
⇕      ADDRESS LIST
 Name: JOE SMITH
→Addr: 123 ANYSTREET
 City: CORVALLIS
 St.  : OR
 Zip  : 97330
 Ph#  : 503-555-1212
```

In the display above, the pointer indicates the currently selected item, and the arrows in the upper-left corner of the display indicate that more data items reside above and below those shown in the display. Each line of the display contains a label (such as "Addr: ") and data (such as "123 ANYSTREET"). The menu keys have been defined by the input parameters.

The input parameters to the Data Browser control the appearance of the data and the options available to the user. For instance, by omitting the title bar and specifying a small font, many data items can be shown in the display at once:



```
 21.45
 77.32
 31.78
→98.45
 123.9987
 4.21
 69.77
 .10042
 7.95
```

Input Parameters

The input parameters to the Data Browser are four lists:

Level 4: { *label list* }

This list contains the label objects. Long labels will be truncated to 25 characters. An empty list may be supplied if no labels are desired.

Level 3: { *data list* }

This list contains the data objects, and must contain at least one object. The data list and label list must be the same length.

Level 2: { *menu label list* }

This list contains the objects which will be presented as menu labels. If the label object is an empty string, the menu label will be black and the menu key will generate an error beep when pressed. If the label object is the string "NULLKEY", the menu label will be white and the menu key will generate an error beep when pressed. A 21x8 graphics object may be used for the key label (see *The Menu Label Builder*). An empty list is acceptable, but the display will still show black labels.

Level 1: { *font first_item current_item edit_flag title* }

The *font* is specified by a real number: 1 for the small font (3x5), or 2 for the medium font (5x7). The real number *first_item* specifies the index of the first data item displayed. The real number *current_item* specifies which data item will be pointed to by the pointer. If *first_item* specifies a row that would force the last data item to appear above the bottom of the display, the value is adjusted to place the last data item at the bottom of the display. If the pointer is off-screen relative to the *first_item*, the data is positioned to place the pointer in the display. If the real number *edit_flag* is non-zero, the user may edit the data items. If *edit_flag*<0, no type checking will be performed. The *title* is specified by a string. Long title strings will be truncated to 19 characters. If an empty string is supplied, the top of the display will be used to present additional data and the arrows indicating data beyond the boundaries of the display will not appear.

Output Parameters

The results from the Data Browser are either three or four items, depending on the original state of the *edit_flag*:

Level 4: { *data list* }
This list contains the data objects (which may have been edited). The data list will not be returned if *edit_flag* was zero.

Level 3: { *font first_item current_item edit_flag title* }
This list is similar to the level 1 input list. The real number *first_item* is the index in the data list of the first data item displayed when the Data Browser terminated. The real number *current_item* is the index of the data item at the pointer when the Data Browser terminated. The *font*, *edit_flag*, and *title* are the same as the input parameter.

Level 2: *current_item*
The real number *current_item* is the index of the data item at the pointer when the Data Browser terminated.

Level 1: *terminator_key*
The *terminator_key* indicates how the user terminated the Data Browser:

0: Zero is returned when the user presses ATTN.

1: One is returned when the user presses ENTER.

−n: If the result is a negative number, the absolute value indicates which menu key was pressed.

Active Keys

While the Data Browser is running, the following keys are active:

[▲] [▼] The arrow keys may be used to move the pointer. Press [↰] and an arrow key to move the pointer one screen at a time. Press [↱] and an arrow key to move to the ends of the catalog.

[↱] [VISIT] If a data item cannot fit within the display (indicated by ...), the [VISIT] key displays as much of the item as will fit in the display, up to 154 characters. Pressing [ATTN] or [ENTER] will return to the original Data Browser display. If the data item fits in entirely in the display, [↱] [VISIT] will generate an error beep. See *Viewing Large Data Items* below.

[↰] [EDIT] If the *edit_flag* is non-zero, pressing [↰] [EDIT] presents a line editor for the current data item. The edit session can be aborted by pressing [ATTN], or accepted by pressing [ENTER]. The input supplied by the user is checked for proper syntax to confirm that a legitimate object results. See *Editing Data Items* below.

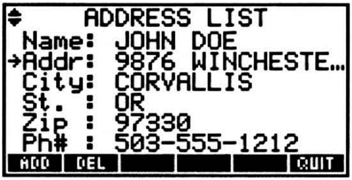
[α] Pressing [α] produces a prompt for a search string. The data list will be searched for a data item containing the search string starting at the next item and wrapping around if necessary. The search is case-sensitive. See *Searching for Data* below.

MENU Pressing a non-null menu key will terminate the Data Browser, indicating which menu key was pressed and the location of the pointer.

[ENTER] Terminates the Data Browser, indicating the location of the pointer.

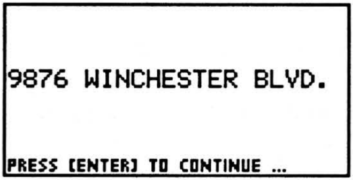
[ATTN] Terminates the Data Browser.

Viewing Large Data Items. The Data Browser has a facility for viewing data items that are too large to fit within a line on the display. For example, consider the display below:



```
⇕      ADDRESS LIST
 Name: JOHN DOE
→Addr: 9876 WINCHESTE…
 City: CORVALLIS
 St. : OR
 Zip : 97330
 Ph# : 503-555-1212
```

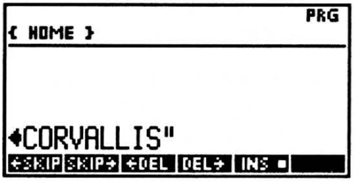
The current data item will not fit in the display, as indicated by the ellipses (...) at the end of the line. Pressing ↱ VISIT produces a full-screen display showing up to 154 characters:



```
9876 WINCHESTER BLVD.

PRESS [ENTER] TO CONTINUE ...
```

Pressing ATTN or ENTER will return to the original Data Browser display.

Editing Data Items. If the *edit_flag* is non-zero, pressing ↰ EDIT presents the command line editor:

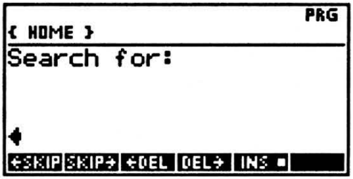


```
                                PRG
{ HOME }

◂CORVALLIS"
```

The menu keys are identical to the command line editor, but the stack is not available. User key definitions and HP 48 menus may be used. The edit session can be aborted by pressing ATTN, or the new data can be accepted by pressing ENTER. There are two important points to consider about editing data items:

- The new data is checked for proper syntax, and must result in a legitimate object. For instance, if the new data represents a program, it must be syntactically correct. String data objects must be surrounded by quotes.
- If *edit_flag* is negative, no type-checking occurs. If the results of the browser session are destined for a type-dependent procedure, such as filling a numeric array, it may be wise set *edit_flag* positive to check the user's input.

Searching for Data. Pressing [α] produces a prompt for a search string:



```
                         PRG
{ HOME }
Search for:


◆
←SKIP SKIP→ ←DEL DEL→ INS ■
```

The menu keys are identical to the command line editor, but the stack is not available. User key definitions and HP 48 menus may be used. The search prompt can be aborted by pressing [ATTN], or the search string can be accepted by pressing [ENTER].

The search begins just past the current data item, and wraps around if necessary. The search ends at the first data item found that contains the search string. Labels are ignored during the search.

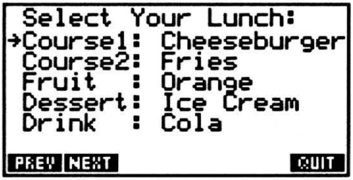
Example: The "address list" example on the first page of this chapter was illustrated using the following program:

NAMES (308 bytes, checksum #2C49h)

```
«
  {
    "Mbr#: " "Name: " "Addr: " "City: "
    "St. : " "Zip : " "Ph# : " "Date: "
  }
  {
    47
    "JOE SMITH"
    "123 ANYSTREET"
    "CORVALLIS"
    "OR"
    97330
    "503-555-1212"
    11.241989
  }
  { "ADD" "DEL" "" "" "" "QUIT" }
  { 2 2 3 1 "    ADDRESS LIST" }
  STD DBR
»
```

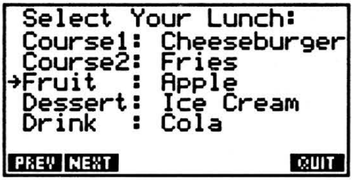
Note that in this example, the first row has been set to two and current row has been set to three, so that the name appears first at the top of the display and the pointer is set to the address line.

Example: The program LUNCH on the next page illustrates a small application that uses the Data Browser and four Tool Library commands: EXTRACT, NXTOB, PRVOB, and REPLACE.



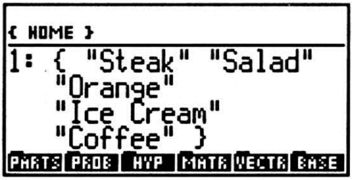
```
 Select Your Lunch:
→Course1: Cheeseburger
 Course2: Fries
 Fruit  : Orange
 Dessert: Ice Cream
 Drink  : Cola
```

While the program LUNCH is running, the PREV and NEXT keys may be used to change the selection for each of five categories. For instance, pressing NEXT with the pointer on the "Fruit" line selects the next available fruit selection:



```
 Select Your Lunch:
 Course1: Cheeseburger
 Course2: Fries
→Fruit  : Apple
 Dessert: Ice Cream
 Drink  : Cola
```

The program is terminated by pressing either QUIT, ENTER, or ATTN. The selections are returned in a list:



```
{ HOME }
1: { "Steak" "Salad"
    "Orange"
    "Ice Cream"
    "Coffee" }
```

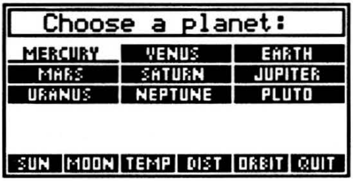
LUNCH (733.5 bytes, checksum #C4ACh)

```
«
  {
    { "Cheeseburger" "Steak" "Chicken" "Hot Dog" }
    { "Fries" "Salad" "Baked Beans" "Corn" }
    { "Orange" "Apple" "Banana" "Pear" }
    { "Ice Cream" "Yogurt" "Cookies" }
    { "Cola" "Coffee" "Milk" "Water" }
  }
  {
   "Course1: " "Course2: " "Dessert : "
   "Fruit   : " "Drink   : "
  }
  OVER LIST→ 1 EXTRACT →LIST 1
  → Choices Labels Lunch Running
  «
     { 2 1 1 0 "Select Your Lunch:" }
     WHILE Running
     REPEAT
       Labels Lunch
       {
        "PREV" "NEXT" "NULLKEY
        "NULLKEY "NULLKEY" "QUIT"
       }
       4 ROLL DBR
       IF
         DUP -6 SAME OVER 0 ≥ OR
       THEN
         3 DROPN Lunch 0 'Running' STO
       ELSE
         Lunch 3 PICK GET Choices 4 ROLL GET
         OVER 4 ROLL
         IF -1 SAME THEN PRVOB ELSE NXTOB END
         Lunch 3 ROLLD REPLACE 'Lunch' STO
       END
     END
  »
»
```

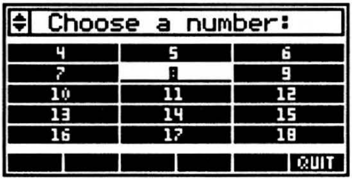
Title Browser

The Title Browser is a utility which provides an efficient method for presenting a series of names or choices to the user with a definable set of menu keys.

The Title Browser appears to the user as three columns of titles with a movable underscore to indicate a choice:



In the display above, there are nine choices available. If there are more than fifteen choices, the title bar will be changed:



The display above shows the order in which the choices are displayed from the input list. The arrows in the upper-left corner of the display indicate that more data items reside above and below those shown in the display.

Input Parameters

The input parameters to the Title Browser are three lists:

Level 3: { *data list* }

This list contains the objects which will be presented as the data. The objects will be converted to a string and centered within the highlighted screen areas. The list must contain at least one object.

Level 2: { *menu label list* }

This list contains the objects which will be presented as menu labels. If the label object is an empty string the menu label will be black and the menu key will generate an error beep when pressed. If the label object is the string "NULLKEY" the menu label will be white and the menu key will generate an error beep when pressed. A 21x8 graphics object may will be used for the key label (see *Menu Label Builder*). An empty list is acceptable, but the display will still show black labels.

Level 1: { *current_item first_row title* }

The real number *current_item* specifies the index in the data list indicated by the underscore. The real number *first_row* specifies the first row of data elements to appear in the display. If *first_row* specifies a row that would force the last row of data to appear above the bottom of the display, the value is adjusted to place the last row of data at the bottom of the display. If the underscore is off-screen relative to the *first_row*, the data is positioned to place the pointer in the display. The *title* is specified by a string. Long titles will be truncated to 21 characters. If there are more then 15 data items, only 20 characters will be displayed, in order to make room for the arrows.

Output Parameters

The results from the Title Browser are three items:

Level 3: { *current_item** *title* }

This list is similar to the level 1 input list. The real number *current_item* is the index in the data list of the underscored data item when the Title Browser terminated. The *title* is the same as the input parameter.

Level 2: *current_item*

The real number *current_item* is the index in the data list of the underscored data item when the Title Browser terminated.

Level 1: *terminator_key*

The *terminator_key* indicates how the user terminated the Title Browser:

0: Zero is returned when the user presses [ATTN].

1: One is returned when the user presses [ENTER].

−n: If the result is a negative number, the absolute value indicates which menu key was pressed.

* current first row

Active Keys

While the Title Browser is running, the following keys are active:

[▲] [▼] [▶] [◀] The arrow keys may be used to move the underscore. Press [↰] [▲] or [↰] [▼] to move the underscore one screen at a time. Press [↱] [▲] or [↱] [▼] to move to the ends of the catalog.

MENU Pressing a non-null menu key will terminate the Title Browser, indicating which menu key was pressed and which item was underscored.

[ENTER] Terminates the Title Browser with a 1, indicating which item was underscored.

[ATTN] Terminates the Title Browser with a 0, indicating which item was underscored.

Example

The "planets" example at the beginning of this chapter was illustrated using the following program:

PLANETS (223 bytes, checksum #4A3Fh):

```
«
  {
    "MERCURY" "VENUS" "EARTH" "MARS" "SATURN"
    "JUPITER" "URANUS" "NEPTUNE" "PLUTO"
  }
  { "SUN" "MOON" "TEMP" "DIST" "ORBIT" "QUIT" }
  { 1 1 "  Choose a planet:   " }
  TBR
»
```

Tool Library

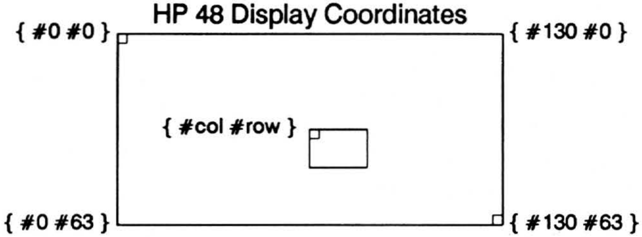
The Tool Library provides 74 new commands that extend the built-in command set of the HP 48. The new commands fall into the following categories (see *Command Index*):

- *Array Operations.* Ten commands facilitate the addition, deletion, exchange, or replacement of rows and columns in an array.
- *Graphics.* Eight commands provide pixel manipulation for graphics objects on the stack, coordinate conversions, and graphics object rotation.
- *List Manipulation.* Twelve commands perform list decomposition, manipulation, and sorting.
- *Meta-Object Utilities.* Fourteen commands provide tools for manipulating meta-objects.
- *Set Utilities.* Six commands manipulate lists as sets.
- *Stack Manipulation.* Seven commands perform stack movement and sorting.
- *String Manipulation.* Twenty-two commands perform extensive string manipulations.
- *Other Commands.* Two commands calculate the day of the week or the day of the year given a date. Two commands extract variable names from a program or equation and search user memory for variables by name or type. The XTIME command calculates execution times.

Graphics

The graphics commands in the Tool Library use pixel coordinates to identify a pixel in a graphics object. A pixel coordinate consists of a list containing two binary integers, { *#col #row* }.

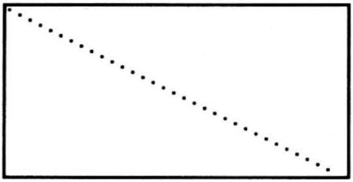
The upper-left pixel of a graphics object is represented by { #0 #0 }. Graphics objects placed into or extracted from *PICT* with the built-in commands GOR, GXOR, SUB, or REPL are located by their upper-left corner. Similarly, the Tool Library commands PXOFF, PXON, and PX? assume that the upper-left pixel of a graphics object is { #0 #0 }.


HP 48 Display Coordinates

The built-in commands PX→C and C→PX convert between user coordinates, such as (x,y), and pixel coordinates. The Tool Library commands PX→R and R→PX simplify the translation between pixel coordinates and loop indices or calculation results.

Example: The following program fragment (64.5 bytes, checksum #6331h) draws a dotted line in *PICT*. The command R→PX is used to form the pixel coordinate for PIXON.

```
«
 0 62 FOR i i 2 * i R→PX PIXON 2 STEP { } PVIEW
»
```



Set Utilities

The set utilities assume (but do not require) that a set is defined as a list of unique objects. In combination with other commands, the set utilities can simplify some otherwise complicated procedures.

A set on the stack may be as simple as an empty list, or a list of many different objects. The command →SET may be used to ensure that all the objects in a set are unique. The command ADJOIN adds an object to a list only if the object does not appear in the list.

The commands DIFF, INTERSECT, SDIFF, and UNION perform set operations or comparisons.

Example: Variables From Equations. The following program fragment (49 bytes, checksum #81A2h) uses the set utility UNION and the command EQNVARS to return a list of all the variables used by a list of equations:

```
«
  OBJ→ MREVERSE { }
  1 ROT
  START
    SWAP EQNVARS DROP UNION
  NEXT
»
```

In the example above, the UNION command is used to ensure that the variables found are added to the output list only if they are unique. The MREVERSE command is used here to reverse the order of the equations on the stack so the variables in the output list appear in the left-to-right order that you would read them in the input list.

Example: Finding Variables Containing Real Numbers.
A complex directory structure can lead to confusion: where is a variable *X* which does not contain a real number? The following program fragment (43.5 bytes, checksum #5584h) uses the set utility INTERSECT and the command VFIND to return a list of all the variables named *X* that contain a real number:

```
«
   'X' VFIND →LIST 0 VFIND →LIST INTERSECT
»
```

In the example above, each call to the VFIND command returns a list of paths. The INTERSECT command is used to ensure that only variables that *do* contain real numbers are returned.

Example: Finding Variables NOT Containing Real Numbers.
The following program fragment (43.5 bytes, checksum #96A9h) uses the set utility DIFF and the command VFIND to return a list of all the variables named *X* that do not contain a real number:

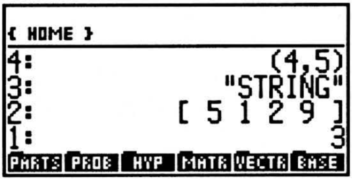
```
«
   'X' VFIND →LIST 0 VFIND →LIST DIFF
»
```

In the example above, each call to the VFIND command returns a list of paths. The DIFF command is used to ensure that only variables that *do not* contain real numbers are returned.

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 will be instances when manipulating groups objects on the stack is more efficient than keeping the objects in lists. The meta-object utilities presented below condense the stack operations into efficient system-code.

The following display shows a meta-object consisting of three objects and their count:



```
{ HOME }
4:              (4,5)
3:           "STRING"
2:        [ 5 1 2 9 ]
1:                  3
```

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 complex number (3,4) in position 1, and meta-objects in positions 2 and 3:

```
"MARS" "JUPITER" 2      2 19 69 3      (3,4)
_____________________   ___________   _______  →
     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. The meta-object command 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.

The meta-object utilities, described in the command reference, consist of the following commands:

MAH	Adds an object to the head of a meta-obj in position 1
MAH2	Adds an object to the head of a meta-obj in position 2
MAM2	Concatenates two meta-objs
MAT	Adds an object to the tail of a meta-obj in position 1
MAT2	Adds an object to the tail of a meta-obj in position 2
MDH	Extracts an element from the head of a meta-obj in pos. 1
MDH2	Extracts an element from the head of a meta-obj in pos. 2
MDT	Extracts an element from the tail of a meta-obj in pos. 1
MDT2	Extracts an element from the tail of a meta-obj in pos. 2
ML→M	Converts lists in positions 1 and 2 into meta-objs
MM→L	Converts meta-objs in positions 1 and 2 into lists
MREVERSE	Reverses the order of the objects in a meta-obj
MSWAP	Swaps the meta-objs in positions 1 and 2
MZ2	Places an empty meta-obj in meta-stack position 2

Other commands in the Tool Library that accept or return parameters in the form of meta-objects are:

EXTRACT	Returns the *m*th element from *n* lists
LSORT	Sorts a series of *n* lists based on the *m*th element
QSORT	Sorts a series of objects
VFIND	Finds variables in user memory
→WORDS	Separates a string into individual words

Example: Testing Variables. If the variables used by a program or equation depend upon the initial conditions of certain variables, a program to show which variables exist in the current path may be helpful.

The following program expects an equation or program as input and returns lists indicating which global variables are defined and undefined. The program uses the meta-object utilities MZ2, MDT, MAH, MAH2, and MM→L. The undefined variables are kept in position 1, and the defined variables are moved to the meta-object in position 2.

ESCAN (161 bytes, checksum #8B5Dh)

```
«
  EQNVARS DROP DTAG           Get global variables
  OBJ→                        Explode list for count
  IF DUP THEN                 Process if there are some global vars
    MZ2 DUP 1 SWAP
    START
      MDT DUP VTYPE
      IF -1 SAME              Does variable exist?
      THEN                    If nonexistent,
        MAH                     and add to "undefined" meta-obj.
      ELSE
        MAH2                  If exists, add to "defined" meta-obj.
      END
    NEXT MM→L                 Convert meta-obs to lists.
  ELSE
    DROP                      If there were no global variables,
    { } { }                     return two empty lists.
  END
  "Undefined" →TAG SWAP       Add tags.
  "Defined" →TAG
»
```

Temporary Memory

A large part of the motivation for using meta-objects has to do with the use of temporary memory in the HP 48. The 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 performed. Enter the numbers 1.5 and 2.6 on the stack. These numbers now reside in temporary memory, referred to by pointers on the 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 stack. The objects 1.5 and 2.6 remain in temporary memory, referenced by pointers that point to the Last Arguments.

Now add 2.8 to the result in level 1. The level 1 pointer on the 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 stack. In general, the system will perform an automatic NEWOB where it makes 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 three situations in which NEWOB can be used explicitly for better control of temporary memory usage:

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

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

  After executing this program, level 1 of the 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 referenced as one of the GET command's arguments.

- Recalling an object to the stack places a pointer to the object on the stack. In the case of backup objects in a port, which consist of an object, name, and checksum combined into a single object, recalling it to the stack places a pointer to the object *within* the backup object on the stack. This is why the system does not do an automatic NEWOB. To purge a backup object from a port while retaining a copy in temporary memory, recall it and execute NEWOB. Then the backup object may be purged because there are no references to it.

- The commands PXON and PXOFF in the Tool Library modify the graphics object directly without creating a copy. If there are several pointers on the stack to a graphics object modified by PXON or PXOFF, each of those pointers will point to the changed graphics object in memory. The NEWOB command may be used in this situation to ensure there are no other references to the graphics object being changed.

Command Index

This index lists the commands in the Tool Library, grouped into subject areas. Some commands or functions appear more than once.

ARRAY OPERATIONS

DELCOL	Deletes a column from an array
DELROW	Deletes a row from an array
EXCOL	Exchanges two columns in an array
EXROW	Exchanges two rows in an array
GETCOL	Extracts a column from an array
GETROW	Extracts a row from an array
INSCOL	Inserts a column into an array
INSROW	Inserts a row into an array
PUTCOL	Replaces a column in an array
PUTROW	Replaces a row in an array

GRAPHICS

PX+	Adds two graphics pixel coordinates
PX−	Subtracts two graphics pixel coordinates
PXOFF	Clears a pixel in an arbitrary graphics object
PXON	Sets a pixel in an arbitrary graphics object
PX?	Tests a pixel in an arbitrary graphics object
PX→R	Converts pixel coordinates into two real numbers
ROTATE	Rotates a graphics object
R→PX	Converts two real numbers into pixel coordinates

LIST MANIPULATION

CAR	Returns the first object of a list
CDR	Returns a list minus its first object
CUT	Splits a list into the first and remaining objects
EXTRACT	Returns the *m*th element from each of a series of lists
LSORT	Sorts a series of lists based on the *m*th element
NXTOB	Returns the next choice from a list of choices
PRVOB	Returns the previous choice from a list of choices
SPLIT	Splits a list into two lists
REPLACE	Replaces all occurrences of an object in a list
REVERSE	Reverses the order of objects in a list
ROTATE	Rotates the objects in a list
→SET	Removes duplicate objects from a list

META-OBJECT UTILITIES

MAH	Adds an object to the head of a meta-obj in position 1
MAH2	Adds an object to the head of a meta-obj in position 2
MAM2	Concatenates two meta-objs
MAT	Adds an object to the tail of a meta-obj in position 1
MAT2	Adds an object to the tail of a meta-obj in position 2
MDH	Extracts an element from the head of a meta-obj in pos. 1
MDH2	Extracts an element from the head of a meta-obj in pos. 2
MDT	Extracts an element from the tail of a meta-obj in pos. 1
MDT2	Extracts an element from the tail of a meta-obj in pos. 2
ML→M	Converts in lists positions 1 and 2 into meta-objs
MM→L	Converts meta-obs in positions 1 and 2 into lists
MREVERSE	Reverses the order of the objects in a meta-obj
MSWAP	Swaps the meta-objs in positions 1 and 2
MZ2	Places an empty meta-obj in meta-stack position 2

SET UTILITIES

ADJOIN	Adds an object to a list if it is unique
DIFF	Returns the set difference of two lists
INTERSECT	Returns the set intersection between two lists
SDIFF	Returns the set symmetric difference of two lists
→SET	Removes duplicate objects from a list
UNION	Returns the set union of two lists

STACK MANIPULATION

KEEP	Keeps the bottom *n* objects on the stack
MREVERSE	Reverses the order of the first *n* stack objects
NDUP	Creates *n* copies of an object
QSORT	Sorts *n* objects on the stack
SRLL	Rotates *n* objects on the stack up *m* times
SRLLD	Rotates *n* objects on the stack down *m* times
SXCH	Exchanges objects at levels *m* and *n*

STRING MANIPULATION

CAR	Returns the first character of a string
CDR	Returns a string minus its first character
CUT	Splits a string into the first and remaining characters
ICAPS	Converts the words in a string to initial caps
LCASE	Converts the characters in a string to lowercase
LTRIM	Removes leading spaces and tabs from a string
PUTCHR	Places character code *n* in a string
REPLACE	Replaces all occurrences of a substring in a string
REVERSE	Reverses the order of characters in a string
ROTATE	Rotates the characters in a string
RPTSTR	Creates a string of *n* substrings
RTRIM	Removes trailing spaces and tabs from a string
SPLIT	Divides a string into two strings
→STDSTR	Converts an object to a string in standard display mode
STRCON	Rapid creation of new character strings
STRCTR	Centers a string in a specified number of spaces
SUBNUM	Returns the character code of a string's *n*th character
→TIO	Converts a string to its translated form for I/O
TIO→	Converts a string from its translated form for I/O
TRIM	Removes leading and trailing spaces and tabs from a string
UCASE	Converts the characters in a string to uppercase
→WORDS	Separates a string into individual words

OTHER COMMANDS

DOW	Returns the day of the week given a date
DOY	Returns the day of the year given a date
EQNVARS	Returns a list of global variables in an equation or program
VFIND	Find all occurrences of a variable or object type in user memory
XTIME	Calculates execution times

Error Messages

The Tool Library contains four new error messages:

Hex	Dec	Error Message
30801	198657	Invalid Pos 1 Meta – Obj
30802	198658	Invalid Pos 2 Meta – Obj
30803	198659	Empty Meta – Obj
30804	198660	Inconsistent Data

Command Reference

This command reference lists the stack diagrams for each of the commands in the Tool Library. Each entry lists the name, description, and stack diagrams. An example is provided to show how each command works.

NAME

		Input			*Output*	
Level_3	Level_2	Level_1	→	Level_3	Level_2	Level_1

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
(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
"string"	Character string
"chr"	Character string containing only one character
{list}	List of objects
grob	Graphics object
{ #x #y }	Pixel coordinates
date	Date in current date format
meta	Meta-object (see *Meta-Objects*)
type	Object type (see *Object Types*)
[vector]	Real or complex vector
[[matrix]]	Real or complex matrix
'global'	Global name
T/F	Test result: 0 (false) or non-zero (true)

Meta-object utilities are described with a notation presented in *Meta-Objects*.

ADJOIN

XLIB 803000, 776 0

Adds an object to a list if the object is not a member of the list.

ADJOIN

{list$_1$} obj → {list$_2$}

Examples:

```
{ 11 22 33 } 33   →   { 11 22 33 }

{ 11 22 33 } 44   →   { 11 22 33 44 }
```

Related Commands: DIFF, INTERSECT, SDIFF, →SET, UNION

CAR

XLIB 803100, 776 1

The command CAR may be used to extract the first element of a list or the first character from a string. When a list object is extracted, a NEWOB is performed to free the element from the list (see *Temporary Memory*).

CAR

" "	→	" "
"string_1"	→	"string_2"
{ }	→	{ }
{obj_1 ... obj_n}	→	obj_1

Examples:

```
        "ABCD"   →   "A"

 { 3 9 8 2 }   →   3
```

Related Commands: CDR, CUT, EXTRACT, NXTOB, PRVOB, SPLIT, REPLACE, REVERSE, ROTATE, →SET

CDR

XLIB 803200, 7762

The command CDR may be used to remove the first object from a list or the first character from a string.

CDR

" "	→	" "
"string$_1$"	→	"string$_2$"
{ }	→	{ }
{obj$_1$ ... obj$_n$}	→	{obj$_2$ ... obj$_n$}

Examples:

```
     "ABCD"   →   "BCD"

{ 3 9 8 2 }   →   { 9 8 2 }
```

Related Commands: CAR, CUT, EXTRACT, NXTOB, PRVOB, SPLIT, REPLACE, REVERSE, ROTATE, →SET

CUT

X27B 803300, 776 3

The command CUT may be used to split a list or string into the first and remaining components.

When a list object is extracted, a NEWOB is performed to free the element from the list (see *Temporary Memory*).

CUT

" "	→	" " " "
"string$_1$"	→	"string$_2$" "chr"
{ }	→	{ } { }
{obj$_1$ ... obj$_n$}	→	{obj$_2$ ... obj$_n$} obj$_1$

Examples:

```
      "ABCD"  →  "BCD" "A"

{ 3 9 8 2 }   →  { 9 8 2 }  3
```

Related Commands: CAR, CDR, EXTRACT, NXTOB, PRVOB, SPLIT, REPLACE, REVERSE, ROTATE, →SET

DELCOL

XLIB 803400, 776 4

The command DELCOL may be used to delete a column from a vector or matrix. The vector or matrix must have at least two columns.

DELCOL

[vector$_1$]	n	→	[vector$_2$]
[[matrix$_1$]]	n	→	[[matrix$_2$]]

Examples:

```
        [ 3 9 8 2 ]  3   →   [ 3 9 2 ]

    [[ 11 22 33 ]            [[ 11 33 ]
       44 55 66 ]]  2   →     [ 44 66 ]]
```

Related Commands: DELROW, EXCOL, EXROW, GETCOL, GETROW, INSCOL, INSROW, PUTCOL, PUTROW

DELROW

XLIB 803500, 776 5

The command DELROW may be used to delete a row from a matrix. The matrix must have at least two rows.

DELROW			
[[matrix$_1$]]	n	→	[[matrix$_2$]]

Example:

```
[[ 11 22 33 ]                 [[ 11 22 33 ]
   44 55 66 ]   2   →          [ 77 88 99 ]]
   77 88 99 ]]
```

Related Commands: DELCOL, EXCOL, EXROW, GETCOL, GETROW, INSCOL, INSROW, PUTCOL, PUTROW

776 6

DIFF

Returns the set difference of two lists.

DIFF

{list$_a$} {list$_b$} → {list$_{a\ \mathrm{AND\ NOT}\ b}$}

Examples:

```
  { 1 2 3 4 } { 5 6 }  →  { 1 2 3 4 }

{ 1 2 3 4 } { 3 4 5 }  →  { 1 2 }
```

Related Commands: ADJOIN, INTERSECT, SDIFF, →SET, UNION

"What is in the first set which isn't in the second set?"

NB: NOT vice versa!

776 7

DOW

Returns the day of the week given a date in the current date format. The days are numbered starting with Monday=1, Tuesday=2, etc. The earliest valid date for this function is October 15, 1582.

DOW

date → n

Examples:

```
5.181957   →   6

3.231981   →   1
```

Related Command: DOY

776 8

DOY

Returns the day of the year given a date in the current date format. The earliest valid date for this function is January 1, 1583.

DOY			
	date	→	n

Examples:

```
5.181957   →   138

3.231981   →   82
```

Related Command: DOW

Note: Does not contain the HP-71's DATE BUG.

776 9

EQNVARS

Given a program or equation, EQNVARS returns lists of global and local variables used in the program or equation. If the input to EQNVARS is a global name, the contents of the name must contain an equation or program, and that object will be scanned for variables.

EQNVARS			
« program »	→	Global: { names }	Local: { names }
'equation'	→	Global: { names }	Local: { names }
'name'	→	Global: { names }	Local: { names }

Examples:

```
          '3*4'   →   Global: {   }  Local: {   }

'R=√(X^2+Y^2)'   →   Global: { R X Y }  Local: { }

« → x y « x 2 ^ y 2 ^ + √ 'R' STO » » →
                    Global: { R }  Local: { x y }
```

Note: The built-in Solver in the HP 48 performs a recursive search through variables to find named programs or equations and adds variables found in those objects to the Solve menu. EQNVARS only searches the program or equation itself. Therefore the variables returned by EQNVARS may be a subset of the variables displayed by the Solve menu.

776 10

EXCOL

Exchanges two columns in an array.

EXCOL

[vector$_1$] col$_1$ col$_2$ → [vector$_2$]

[[matrix$_1$]] col$_1$ col$_2$ → [matrix$_2$]]

Examples:

```
      [ 1 2 3 4 ]  2  3  →  [ 1 3 2 4 ]

[[ 1 2 ] [ 3 4 ]]  1  2  →  [[ 2 1 ] [ 4 3 ]]
```

Related Commands: DELCOL, DELROW, EXROW, GETCOL, GETROW, INSCOL, INSROW, PUTCOL, PUTROW

776 11

EXROW

Exchanges two rows in an array.

EXROW

[[matrix$_1$]] row$_1$ row$_2$ → [matrix$_2$]]

Example:

```
[[ 1 2 ] [ 3 4 ]]  1  2  →  [[ 3 4 ] [ 1 2 ]]
```

Related Commands: DELCOL, DELROW, EXCOL, GETCOL, GETROW, INSCOL, INSROW, PUTCOL, PUTROW

776 /12

EXTRACT

The command EXTRACT may be used to return the *m*th element from each of a series of *n* lists. The input and result are formed as meta-objects. A NEWOB is performed to free each element from the list (see *Temporary Memory*).

EXTRACT

{list$_1$} ... {list$_n$} n obj-number → obj$_1$ ... obj$_n$ n

Example:

```
{ 3 91 } { 1 78 } { 8 12 } 3 2   →   91 78 12 3
```

Related Commands: LSORT, MREVERSE

776 13

GETCOL

Returns a column from an array as a matrix consisting of 1-element rows.

```
GETCOL
                    [vector]  col   →    [[column data]]
                  [[matrix]]  col   →    [[column data]]
```

Examples:

```
            [ 1 2 3 ]  2        →   [[ 2 ]]

[[ 1 2 3 ] [ 4 5 6 ]] 2        →   [[ 2 ] [ 5 ]]
```

Related Commands: DELCOL, DELROW, EXCOL, EXROW, GETROW, INSCOL, INSROW, PUTCOL, PUTROW

776 /14

GETROW

Returns a row from an array as a vector.

GETROW			
[vector]	row	→	[row data]
[[matrix]]	row	→	[row data]

Examples:

```
          [ 1 2 3 ]  1   →   [ 1 2 3 ]

[[ 1 2 3 ] [ 4 5 6 ]] 2  →   [ 4 5 6 ]
```

Related Commands: DELCOL, DELROW, EXCOL, EXROW, GETCOL, INSCOL, INSROW, PUTCOL, PUTROW

776 15

ICAPS

Converts the first character of each word in a string to uppercase, and the remaining characters to lowercase. The separation characters are any character code ≤30, 32, and 160.

ICAPS		
"string_1"	→	"string_2"

The case conversion supports the ISO 8859−1 character set in the following ranges:

Lowercase		Uppercase
61h−7Ah	⟷	41h−5Ah
E0h−F6h	⟷	C0h−D6h
F8h−FEh	⟷	D8h−DEh

Examples:

```
      "JOHN SMITH"   →   "John Smith"

"sample sentence"   →   "Sample Sentence"
```

Related Commands: LCASE, UCASE

776 16

INSCOL

The command INSCOL may be used to insert a column into an array. The column number specifies which column will be zero-filled, and may be one greater than the number of columns in the array.

INSCOL

[vector$_1$] n → [vector$_2$]
[[matrix$_1$]] n → [[matrix$_2$]]

Examples:

```
      [ 3 9 8 2 ]  3   →   [ 3 9 0 8 2 ]

  [ (9,4) (8,3) ]  3   →   [ (9,4) (8,3) (0,0) ]

   [[ 11 22 33 ]           [[ 11 0 22 33 ]
      44 55 66 ]   2   →    [ 44 0 55 66 ]
      77 88 99 ]]           [ 77 0 88 99 ]]
```

Related Commands: DELCOL, DELROW, EXCOL, EXROW, GETCOL, GETROW, INSROW, PUTCOL, PUTROW

776 17

INSROW

The command INSROW may be used to insert a row into an array. If the input is a vector (one-dimensional), the result will be a matrix (two-dimensional). The row number specifies which column will be zero-filled, and may be one greater than the number of rows in the array.

INSROW

[vector] n → [[matrix]]
[[matrix$_1$]] n → [[matrix$_2$]]

Examples:

```
                            →   [[ 0 0 0 0 ]
     [ 3 9 8 2 ]   1        →     [ 3 9 8 2 ]]

[[ 11 22 33 ]                   [[ 11 22 33 ]
   44 55 66 ]      4        →     [ 44 55 66 ]
   77 88 99 ]]                    [ 77 88 99 ]
                                  [  0  0  0 ]]
```

Related Commands: DELCOL, DELROW, EXCOL, EXROW, GETCOL, GETROW, INSCOL, PUTCOL, PUTROW

776 18

INTERSECT

Returns the set intersection between two lists.

INTERSECT

{list$_a$} {list$_b$} → {list$_{a\ \mathrm{AND}\ b}$}

Examples:

```
  { 1 2 3 4 } { 5 6 }  →  { }

{ 1 2 3 4 } { 3 4 5 }  →  { 3 4 }
```

Related Commands: ADJOIN, DIFF, SDIFF, →SET, UNION

776 19

KEEP

Keeps the bottom *n* objects on the stack while deleting all objects above *n*.

KEEP

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

Example:

```
"AA" 32 7.1 "B" 2  →  7.1 "B"
```

Related Commands: NDUP, SRLL, SRLLD, SXCH

776 20

LCASE

Converts each character in a string to lowercase.

LCASE

"$string_1$" → "$string_2$"

The case conversion supports the ISO 8859-1 character set in the following ranges:

Lowercase		Uppercase
61h-7Ah	←	41h-5Ah
E0h-F6h	←	C0h-D6h
F8h-FEh	←	D8h-DEh

Example:

```
"SAMPLE SENTENCE"    →    "sample sentence"
```

Related Commands: ICAPS, UCASE

776 21

LSORT

The command LSORT may be used to sort a series of *n* lists based on the *m*th element of each list. The input and result are formed as meta-objects.

The *m*th object in each list must be of the same type and comparable with >. The lists are returned in ascending order (the largest at the bottom of the stack). Use MREVERSE after LSORT to produce a descending order result. The sort order for strings follows the ISO 8859-1 character set (see *Character Codes*).

LSORT

{list_1} ... {list_n} n m → {$\text{list}_?$} ... {$\text{list}_?$} n

Example:

```
{ 3 9 } { 1 7 } { 8 2 } 3 1  →
                             { 1 7 } { 3 9 } { 8 2 } 3
```

Related Commands: EXTRACT, MREVERSE, QSORT

776 22

LTRIM

Removes leading space and tab (#09h) characters from a string.

LTRIM		
"string$_1$"	→	"string$_2$"

Example:

```
"   SAMPLE STRING   "   →   "SAMPLE STRING   "
```

Related Commands: RTRIM, TRIM, →WORDS

776 23

MAH

Adds an object to the head of a meta-object.

MAH

meta_1 obj → meta_2

< ... > obj → < obj ... >

Example:

```
21 32 47 3 99  →  99 21 32 47 4
```

Related Commands: MAH2, MAM2, MAT, MAT2, MDH, MDH2, MDT, MDT2, ML→M, MM→L, MREVERSE, MSWAP, MZ2

776 24

MAH2

Adds an object to the head of a meta-object in position 2.

MAH2

$\text{meta}_2 \quad \text{meta}_1 \quad \text{obj} \quad \rightarrow \quad \text{meta}_2' \quad \text{meta}_1$

$< \text{meta}_2 > \quad < \text{meta}_1 > \quad \text{obj} \quad \rightarrow \quad < \text{obj meta}_2 > \quad < \text{meta}_1 >$

Example:

```
21 32 2 2.3 4.7 2 99   →   99 21 32 3 2.3 4.7 2
```

Related Commands: MAH, MAM2, MAT, MAT2, MDH, MDH2, MDT, MDT2, ML→M, MM→L, MREVERSE, MSWAP, MZ2

776 25

MAM2

Concatenates two meta-objects.

MAM2

meta_2 meta_1 obj → meta_{2+1}

< meta_2 > < meta_1 > → < meta_{2+1} >

Example:

```
21 32 47 3 7.3 4.8 2  →  21 32 47 7.3 4.8 5
```

Related Commands: MAH, MAH2, MAT, MAT2, MDH, MDH2, MDT, MDT2, ML→M, MM→L, MREVERSE, MSWAP, MZ2

776 26

MAT

Adds an object to the tail of a meta-object.

MAT

meta_1 obj → meta_2

< ... > obj → < ... obj >

Example:

```
21 32 47 3 99   →   21 32 47 99 4
```

Related Commands: MAH, MAH2, MAM2, MAT2, MDH, MDH2, MDT, MDT2, ML→M, MM→L, MREVERSE, MSWAP, MZ2

776 27

MAT2

Adds an object to the tail of a meta-object in position 2.

MAT2

$\text{meta}_2 \;\; \text{meta}_1 \;\; \text{obj} \;\; \rightarrow \;\; \text{meta}_2' \;\; \text{meta}_1$

$< \text{meta}_2 > \;\; < \text{meta}_1 > \;\; \text{obj} \;\; \rightarrow \;\; < \text{meta}_2 \;\; \text{obj} > \;\; < \text{meta}_1 >$

Example:

```
21 32 2 2.3 4.7 2 99   →   21 32 99 3 2.3 4.7 2
```

Related Commands: MAH, MAH2, MAM2, MAT, MDH, MDH2, MDT, MDT2, ML→M, MM→L, MREVERSE, MSWAP, MZ2

776 28

MDH

Extracts an object from the head of a meta-object.

MDH

$\text{meta}_1 \rightarrow \text{meta}_2 \text{ obj}$

< obj ... > → < ... > obj

Example:

```
99 21 32 47 4   →   21 32 47 3 99
```

Related Commands: MAH, MAH2, MAM2, MAT, MAT2, MDH2, MDT, MDT2, ML→M, MM→L, MREVERSE, MSWAP, MZ2

776 29

MDH2

Extracts an object from the head of a meta-object in position 2.

MDH2

$\text{meta}_2 \;\; \text{meta}_1 \;\; \rightarrow \;\; \text{meta}_2' \;\; \text{meta}_1 \;\; \text{obj}$

$<\text{obj} \;\; \text{meta}_2> \;\; <\text{meta}_1> \;\; \rightarrow \;\; <\text{meta}_2'> \;\; <\text{meta}_1> \;\; \text{obj}$

Example:

```
99 21 32 3 2.3 4.7 2   →   21 32 2 2.3 4.7 2 99
```

Related Commands: MAH, MAH2, MAM2, MAT, MAT2, MDH, MDT, MDT2, ML→M, MM→L, MREVERSE, MSWAP, MZ2

776 30

MDT

Extracts an object from the tail of a meta-object.

MDT

meta$_1$ → meta$_2$ obj

< ... obj > → < ... > obj

Example:

```
21 32 47 99 4   →   21 32 47 3 99
```

Related Commands: MAH, MAH2, MAM2, MAT, MAT2, MDH, MDH2, MDT2, ML→M, MM→L, MREVERSE, MSWAP, MZ2

776 31

MDT2

Extracts an object from the tail of a meta-object in position 2.

MDT2

$\text{meta}_2 \quad \text{meta}_1 \quad \rightarrow \quad \text{meta}_2' \quad \text{meta}_1 \quad \text{obj}$

$< \text{meta}_2 \ \text{obj} > \quad < \text{meta}_1 > \quad \rightarrow \quad < \text{meta}_2' > \quad < \text{meta}_1 > \quad \text{obj}$

Example:

```
21 32 99 3 2.3 4.7 2  →  21 32 2 2.3 4.7 2 99
```

Related Commands: MAH, MAH2, MAM2, MAT, MAT2, MDH, MDH2, MDT, ML→M, MM→L, MREVERSE, MSWAP, MZ2

776 32

ML→M

Converts two lists into meta-objects.

ML→M

{list$_2$} {list$_1$} → meta$_2$ meta$_1$

{list$_2$} {list$_1$} → < meta$_2$ > < meta$_1$ >

Example:

```
{ 11 22 } { 3.1 4.2 5.1 }  →
                              11 22 2 3.1 4.2 5.1 3
```

Related Commands: MAH, MAH2, MAM2, MAT, MAT2, MDH, MDH2, MDT, MDT2, MM→L, MREVERSE, MSWAP, MZ2

776 33

MM→L

Converts two meta-objects into lists.

MM→L

meta$_2$	meta$_1$	→	{list$_2$}	{list$_1$}
< meta$_2$ >	< meta$_1$ >	→	{list$_2$}	{list$_1$}

Example:

```
11 22 2 3.1 4.2 5.1 3  →
                         { 11 22 } { 3.1 4.2 5.1 }
```

Related Commands: MAH, MAH2, MAM2, MAT, MAT2, MDH, MDH2, MDT, MDT2, ML→M, MREVERSE, MSWAP, MZ2

776 34

MREVERSE

Reverses the order of *n* objects on the stack. This command will reverse the order of 5000 stack items about two seconds.

MREVERSE

$obj_1 \ldots obj_n$ n	→	$obj_n \ldots obj_1$ n
$meta_1$	→	$meta_2$
< obj_1 obj_2 obj_3 >	→	< obj_3 obj_2 obj_1 >

Example:

```
11 22 .2 3.1 2 5  →  2 3.1 .2 22 11 5
```

Related Commands: KEEP, MAH, MAH2, MAM2, MAT, MAT2, MDH, MDH2, MDT, MDT2, ML→M, MM→L, MSWAP, MZ2, NDUP, SRLL, SRLLD, SXCH

776 35

MSWAP

Swaps two meta-objects on the stack.

MSWAP

$\text{meta}_2 \quad \text{meta}_1 \quad \rightarrow \quad \text{meta}_1 \quad \text{meta}_2$

$< \text{meta}_2 > \quad < \text{meta}_1 > \quad \rightarrow \quad < \text{meta}_1 > \quad < \text{meta}_2 >$

Example:

```
11 22 2 3.1 4.2 5.1 3   →   3.1 4.2 5.1 3 11 22 2
```

Related Commands: MAH, MAH2, MAM2, MAT, MAT2, MDH, MDH2, MDT, MDT2, ML→M, MM→L, MREVERSE, MZ2

776 36

MZ2

Places an empty meta-object in meta-stack position 2.

MZ2

meta$_1$ → meta$_{empty}$ meta$_1$

< meta$_1$ > → < > < meta$_1$ >

Example:

```
3.1 4.2 5.1 3  →  0 3.1 4.2 5.1 3
```

Related Commands: MAH, MAH2, MAM2, MAT, MAT2, MDH, MDH2, MDT, MDT2, ML→M, MM→L, MREVERSE, MSWAP

776 37

NDUP

Creates *n* copies of an object on the stack. If *n* is zero, no objects will be returned.

NDUP

obj n → obj ... obj

Examples:

```
  23 0  →
 5.1 3  →  5.1 5.1 5.1
```

Related Commands: KEEP, MREVERSE, SRLL, SRLLD SXCH

saves memory only if n>3

776 38

NXTOB

Given a list of *n* objects and an object, NXTOB finds the location of the object in the list and returns the following object. If the object is found at the end of the list, the first object is returned. If the object is not found in the list, the same object is returned.

NXTOB

$\{obj_1 \ldots obj_n\}\ \ obj_m \quad \rightarrow \quad obj_{m+1}$

Examples:

```
    {  }  "FRED"   →   "FRED"

{ 11 22 33 }  22   →   33

{ 11 22 33 }  33   →   11
```

Related Commands: CDR, CUT, EXTRACT, LSORT, PRVOB, SPLIT, REPLACE, REVERSE, ROTATE, →SET

776 39

PRVOB

Given a list of *n* objects and an object, PRVOB finds the location of the object in the list and returns the previous object. If the object is found at the beginning of the list, the last object is returned. If the object is not found in the list, the same object is returned.

PRVOB

{obj_1 ... obj_n} obj_m → obj_{m-1}

Examples:

```
    {  }  "FRED"   →   "FRED"

{ 11 22 33 }  22   →   11

{ 11 22 33 }  11   →   33
```

Related Commands: CDR, CUT, EXTRACT, LSORT, NXTOB, SPLIT, REPLACE, REVERSE, ROTATE, →SET

776 40

PUTCHR

Places a character at a specified position in a string. The character may be specified by a real number character code or by the first character in a string. In the second instance, PUTCHR is similar to REPL, except that only one character is changed.

PUTCHR

"string$_1$" position code → "string$_2$"
"string$_1$" position "string$_2$" → "string$_3$"

The commands PUTCHR and SUBNUM are designed for applications requiring an index array for values less than 255. Using a string to store the indices as character codes saves considerable memory compared to other storage methods, such as lists or arrays.

Examples:

```
  "JOHN" 3 65    →   "JOAN"

"JOHN" 3 "ABC"   →   "JOAN"
```

Related Commands: STRCON, SUBNUM

776 41

PUTCOL

Replaces a column of data in an array.

PUTCOL

[[matrix$_1$]] col [[new−col]] → [[matrix$_2$]]

Examples:

```
[ 1 2 3 ]  2  [[ 44 ]]  →  [ 1 44 3 ]

[[ 1 2 3 ] [ 4 5 6 ]] 2 [[ 11 ][ 22 ]] →
                                [[ 1 11 3 ] [ 4 22 6 ]]
```

Related Commands: DELCOL, DELROW, EXCOL, EXROW, GETCOL, GETROW, INSCOL, INSROW, PUTROW

776 42

PUTROW

Replaces a row of data in an array.

PUTROW

[vector$_1$] row [new-row] → [vector$_2$]

[[matrix$_1$]] row [new-row] → [[matrix$_2$]]

Examples:

```
[ 1 2 3 ]  1  [ 4 5 6 ]  →  [ 4 5 6 ]

[[ 1 2 3 ] [ 4 5 6 ]] 2 [ 7 8 9 ] →
                                   [[ 1 2 3 ] [ 7 8 9 ]]
```

Related Commands: DELCOL, DELROW, EXCOL, EXROW, GETCOL, GETROW, INSCOL, INSROW, PUTCOL

776 43

PX+

Adds two graphics pixel coordinates.

PX+

$\{ \#_1 \ \ \#_2 \} \ \{ \#_3 \ \ \#_4 \} \ \rightarrow \ \{ \#_{1+3} \ \ \#_{2+4} \}$

Example:

```
{ #3d #7d }   { #6d #1d }   →   { #9d #8d }
```

Related Command: PX−

776 44

PX-

Subtracts two graphics pixel coordinates.

PX-

$$\{\#_1 \quad \#_2\} \ \{\#_3 \quad \#_4\} \quad \rightarrow \quad \{\#_{1-3} \quad \#_{2-4}\}$$

Example:

```
{ #23d #54d }   { #6d #1d }   →   { #17d #53d }
```

Related Command: PX+

776 45

PXOFF

Clears a pixel in an arbitrary graphics object.

PXOFF

grob { $\#_X$ $\#_Y$ } → grob'

Notes:

- This command does not work for *PICT*. Use the command PIXOFF for clearing pixels in *PICT*.
- The upper-left pixel in a graphics object has the coordinate { #0 #0 } (see *Graphics*).
- This command does not return a unique copy of the graphics object. You may wish to execute NEWOB first to ensure that the result is a unique object (See *Temporary Memory*).

Example:

```
GROB 8 2 0304   { #6d #1d }  →  GROB 8 2 0300
```

Related Commands: PXON, PX?

776 46

PXON

Sets a pixel in an arbitrary graphics object.

PXON

grob { #$_X$ #$_Y$ } → grob'

Notes:

- This command does not work for *PICT.* Use the command ~~PIXOFF~~ Pixon for ~~clearing~~ setting pixels in *PICT.*
- The upper-left pixel in a graphics object has the coordinate { #0 #0 } (see *Graphics*).
- This command does not return a unique copy of the graphics object. You may wish to execute NEWOB first to ensure that the result is a unique object (See *Temporary Memory*).

Example:

```
GROB 8 2 0300   { #6d #1d }  →   GROB 8 2 0304
```

Related Commands: PXOFF, PX?

776 47

PX?

Tests a pixel in an arbitrary graphics object.

PX?

grob { $\#_X$ $\#_Y$ } → T/F

Notes:

- This command does not work for *PICT*. Use the command PIX? for testing pixels in *PICT*.
- The upper-left pixel in a graphics object has the coordinate { #0 #0 } (see *Graphics*).

Example:

```
GROB 8 2 0300   { #6d #1d }   →   0
GROB 8 2 0304   { #6d #1d }   →   1
```

Related Commands: PXOFF, PXON

776 48

PX→R

Converts a list of two binary integers to two real numbers.

```
PX→R
                { #col  #row }   →    col  row
```

Example:

```
{ #4d #18d }   →   4   18
```

Related Command: R→PX

776 49

QSORT

The command QSORT may be used to sort a series of *n* objects on the stack. The input and result are formed as meta-objects.

Each object must be of the same type and comparable with >. The objects are returned in ascending order (the largest at the bottom of the stack). Use MREVERSE after QSORT to produce a descending order result. The sort order for strings follows the ISO 8859-1 character set (see *Character Codes*).

QSORT

$\text{obj}_1 \ldots \text{obj}_n \ \ n \ \rightarrow \ \text{obj}_? \ldots \text{obj}_? \ \ n$

Examples:

```
            3 2 8 7 4  →  2 3 7 8 4

"FRED" "ANNE" "ZOE" 3  →  "ANNE" "FRED" "ZOE" 3
```

Related Commands: LSORT, MREVERSE

776 50

REPLACE

The command REPLACE may be used to replace all occurrences of a substring within a string or of objects within a list. String comparisons require an exact match.

REPLACE

"string$_1$" "string$_{search}$" "string$_{repl}$" → "string$_2$"

{list$_1$} obj$_{search}$ obj$_{repl}$ → {list$_2$}

Examples:

```
        "JOHN" "H" "A"   →   "JOAN"

       "ABCBD" "B" "-"   →   "A-C-D"

    { 1 9 3 9 5 } 9 44   →   { 1 44 3 44 5 }

{ (1,1) 2.2 "fred" 44 }  "fred" #33d  →
                             { (1,1) 2.2 #33d 44 }
```

Related Commands: CAR, CDR, CUT, EXTRACT, SPLIT, REVERSE, ROTATE, →SET

776 51

REVERSE

The command REVERSE may be used to reverse the order of characters in a string or objects in a list.

Reversals of large lists will be significantly faster if the list was originally stored in a global variable. The time to reverse a large list is longer than the time required for the MREVERSE command, owing to the overhead of unpacking and re-packing the list objects. Reversing a 1000-element list originating from a global variable should take about three seconds. If the same list originates in temporary memory, the reversal could take several minutes.

String reversals are accomplished at a rate near 12,000 characters per second.

REVERSE

" "	→	" "
"string$_1$"	→	"string$_2$"
{ }	→	{ }
{ obj$_1$ ... obj$_n$ }	→	{ obj$_n$ ... obj$_1$ }

Examples:

```
          "ABCD"     →    "DCBA"

      { 1 2 3 4 5 }  →    { 5 4 3 2 1 }
```

Related Commands: CAR, CDR, CUT, EXTRACT, LSORT, SPLIT, REPLACE, ROTATE, →SET , MREVERSE

776 52

ROTATE

The command ROTATE may be used to rotate the contents of a list, string, or graphics object. The direction of rotation is controlled by the sign of x:

$x<0$	Rotates left	COUNTERCLOCKWISE ↺
$x=0$	No change	
$x>0$	Rotates right	CLOCKWISE ↻

Graphics objects are rotated 90° to the left for $x<0$, or 90° to the right for $x>0$. If $|x|$ is greater than the length of the list or string, the rotation count will be calculated MOD the list or string size.

ROTATE

"string_1"	x	→	"string_2"
list_1	x	→	list_2
grob_1	x	→	grob_2

String Examples:

```
          ""     5   →   ""

     "ABCDE"    2   →   "DEABC"

     "ABCDE"   -2   →   "CDEAB"
```

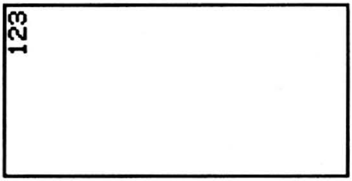
List Examples:

```
        {  }        5   →   {  }

 { 1 2 3 4 5 }  2   →   { 4 5 1 2 3 }

 { 1 2 3 4 5 } -2   →   { 3 4 5 1 2 }
```

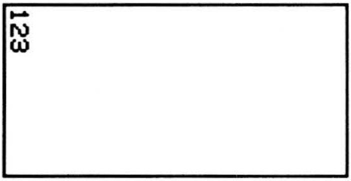
Graphics Examples:

```
Graphic 21 x 8 -1   →   Graphic 8 x 21
Graphic 21 x 8  0   →   Graphic 21 x 8
Graphic 21 x 8  1   →   Graphic 8 x 21
```

```
« "123" 2 →GROB -1 ROTATE PICT STO { } PVIEW »
```



```
« "123" 2 →GROB 1 ROTATE PICT STO { } PVIEW »
```



Note: Rotation performance for graphics objects is reasonable for small objects, such as axis labels for graphs, however the algorithm for rotating graphics was optimized for space as opposed to speed. Consequently, rotating a 131x64 graphics object takes just under 15 seconds. The rotation requires enough free memory to construct a second temporary graphics object.

Related Commands: CAR, CDR, CUT, EXTRACT, LSORT, NXTOB, PRVOB, SPLIT, REPLACE, REVERSE, →SET

776 53

RPTSTR

Creates a string consisting of *n* repetitions of an input string. If only one character is to be repeated, the STRCON command will give faster performance.

RPTSTR

"string" n → "string ... string"

Examples:

```
"ABC" 0   →   ""

"ABC" 3   →   "ABCABCABC"
```

Related Command: STRCON

776 54

RTRIM

Removes trailing space and tab (#09h) characters from a string.

RTRIM

"string$_1$" → "string$_2$"

Example:

```
"   SAMPLE STRING   "  →  "   SAMPLE STRING"
```

Related Commands: LTRIM, TRIM, →WORDS

776 55

R→PX

Converts two real numbers to a list of two binary integers.

R→PX			
	col row	→	{ #col #row }

Example:

```
45  37   →   { #45d #37d }
```

Related Command: PX→R

776 56

SDIFF

Returns the set symmetric difference of two lists.

SDIFF

$\{list_a\}$ $\{list_b\}$ → $\{list_{a\ XOR\ b}\}$

Examples:

```
  { 1 2 3 4 } { 5 6 }  →  { 1 2 3 4 5 6 }

{ 1 2 3 4 } { 3 4 5 }  →  { 1 2 5 }
```

Related Commands: ADJOIN, DIFF, INTERSECT, →SET, UNION

"What Is In One Set Or The Other
But Not Both?" (XOR)

776 57

→SET

Removes duplicate objects from a list.

→SET		
$\{\text{list}_1\}$	→	$\{\text{list}_2\}$

Examples:

```
  { 1 2 3 4 }   →   { 1 2 3 4 }

{ 4 1 2 3 2 1 }   →   { 4 1 2 3 }
```

Related Commands: ADJOIN, INTERSECT, CAR, CDR, CUT, DIFF, EXTRACT, NXTOB, PRVOB, SDIFF, SPLIT, REPLACE, REVERSE, ROTATE, UNION

776 58

SPLIT

The command SPLIT may be used to divide a list or string into first *m* and remaining components.

SPLIT

" "	m	→	" " " "
"string_1"	m	→	"string_2" "string_3"
{ }	m	→	{ } { }
{obj_1 ... obj_n}	m	→	{obj_{m+1} ... obj_n} {obj_1 ... obj_m}

Examples:

```
          "ABCDE"  0  →  "ABCDE"  ""

          "ABCDE"  3  →  "DE"  "ABC"

              { }  3  →  { }  { }

    { 3 9 8 2 7 }  0  →  { 3 9 8 2 7 }  { }

    { 3 9 8 2 7 }  2  →  { 8 2 7 } { 3 9 }
```

Related Commands: CAR, CDR, CUT, REVERSE, ROTATE

776 59

SRLL

Rotates *n* objects on the up stack *m* times.

SRLL			
	$obj_1 \ldots obj_n$ n m	→	$obj_{n-m+1} \ldots obj_n$ $obj_1 \ldots obj_{m+1}$

Example:

```
11 22 33 44 55 5  2   →   ~~44 55 11 22 33~~
                          33 44 55 11 22
```

Related Commands: KEEP, MREVERSE, NDUP, SRLLD, SXCH

776 60

SRLLD

Rotates *n* objects on the stack down *m* times.

SRLL

$obj_1 \ldots obj_n$ n m → $obj_{m+1} \ldots obj_n$ $obj_1 \ldots obj_m$

Example:

11 22 33 44 55 5 2 → ~~33 44 55 11 22~~

44 55 11 22 33

Related Commands: KEEP, MREVERSE, NDUP, SRLL, SXCH

776 61

→STDSTR

Converts an object to a string (like →STR), using STD display mode and a wordsize of 64 bits.

→STDSTR		
obj	→	"string"

Examples:

Assuming the current display mode is 2 FIX, execute 'π' →NUM, then →STDSTR:

```
3.14   →   "3.14159265359"
```

Assuming the current wordsize is 8 and HEX mode is set, enter # 123h. The wordsize of 8 causes the binary integer to be displayed as # 23h. To see the full value, execute →STDSTR:

```
# 23h   →   "# 123h"
```

776 62

STRCON

Creates a string consisting of *n* repetitions of a character *code*. Strings are created at a rate nearing 20,000 characters per second.

STRCON

code n → "string"

Examples:

```
 65 0   →   ""

65 10   →   "AAAAAAAAAA"
```

Related Commands: PUTCHR, RPTSTR, SUBNUM

776 63

STRCTR

Centers a string in a specified number of spaces. If the number of spaces added is not even, the extra space will be added to the end of the string.

STRCTR

"string$_1$" n → "string$_2$"

Example:

```
 "SAMPLE"   9   →   " SAMPLE  "

"SAMPLE"   10   →   "  SAMPLE  "
```

Related Command: TRIM

776 64

SUBNUM

Returns the character code of the *n*th character of a string.

```
SUBNUM
                    "string"  n  →    code
```

The commands PUTCHR and SUBNUM are designed for applications requiring an index array for values less than 255. Using a string to store the indices as character codes saves considerable memory compared to other storage methods, such as lists or arrays.

Example:

```
"ALPHABET"  4   →   72
```

Related Commands: PUTCHR, STRCON

776 65

SXCH

Exchanges objects at levels *m* and *n* on the stack.

SXCH

... obj_m ... obj_n ... m n → ... obj_n ... obj_m ...

Example:

```
58 22 87 34 14 4 2  →  58 34 87 22 14
```

Related Commands: KEEP, MREVERSE, NDUP, QSORT, SRLL, SRLLD

776 66

→TIO

Converts a string to its translated form for output, respecting the current TRANSIO setting in *IOPAR*. If there is no *IOPAR* in the HOME directory, a new one will be created in the HOME directory with the default TRANSIO setting of 1 (see *Character Translations*).

→TIO		
"string$_1$"	→	"string$_2$"

Example:

```
"« → x « x SIN x / » »" →
              "\<< \-> r \<< x SIN x / \>> \>>"
```

Related Command: TIO→

776 67

TIO→

Converts a string from its translated form for output, respecting the current TRANSIO setting in *IOPAR*. If there is no *IOPAR* in the HOME directory, a new one will be created in the HOME directory with the default TRANSIO setting of 1 (see *Character Translations*).

TIO→			
	"string$_1$"	→	"string$_2$"

Example:

```
"\<< \-> r \<< x SIN x / \>> \>>" →
                                  "« → x « x SIN x / » »"
```

Related Command: →TIO

776 68

TRIM

Removes leading and trailing space and tab (#09h) characters from a string.

TRIM

"string$_1$" → "string$_2$"

Example:

```
"   SAMPLE STRING   "  →  "SAMPLE STRING"
```

Related Commands: LTRIM, RTRIM, STRCTR, →WORDS

776 69

UCASE

Converts each character in a string to uppercase.

UCASE

"string$_1$" → "string$_2$"

The case conversion supports the ISO 8859-1 character set in the following ranges:

	Lowercase		Uppercase	
97-122	61h-7Ah	→	41h-5Ah	65-90
224-246	E0h-F6h	→	C0h-D6h	192-214
248-254	F8h-FEh	→	D8h-DEh	216-222

Example:

```
"sample sentence"   →   "SAMPLE SENTENCE"
```

Related Commands: ICAPS, LCASE

776 70

UNION

Returns the set union of two lists.

UNION

$\{list_a\}$ $\{list_b\}$ → $\{list_{a\ OR\ b}\}$

Examples:

```
  { 1 2 3 4 } { 5 6 }  →  { 1 2 3 4 5 6 }
{ 1 2 3 4 } { 3 4 5 }  →  { 1 2 3 4 5 }
```

Related Commands: ADJOIN, DIFF, INTERSECT, SDIFF, →SET

776 71

VFIND

Given an global variable name or an object type, VFIND performs a recursive search for a global variables starting at HOME and returns a series of paths (each of which is a list) to each occurrence of a variable in user memory meeting the search criteria (see *Object Types*).

VFIND

name → {path$_1$} ... {path$_m$} n

type → {path$_1$} ... {path$_m$} n

Examples:

```
    4   →   0

    0   →   { HOME X } { HOME REALS Y } 2

  'X'   →   { HOME X } { HOME REALS X } 2
```

776 72

→WORDS

Separates a string into words and their count. The separation characters are any character code ≤30, 32, and 160. Adjacent separator characters are treated as a single separator character.

→WORDS

"word$_1$ ... word$_n$" → "word$_1$" ... "word$_n$" n

Examples:

```
           ""   →   0

"A  TEST STRING"   →   "A" "TEST" "STRING" 3
```

Related Commands: LTRIM, RTRIM, TRIM

776 73

XTIME

Times the execution time for an object such as a command or program. An initial garbage collection is performed (see *Temporary Memory*) to produce the most reliable result, and the result is rounded to the nearest thousandth of a second.

XTIME

object → seconds

Example:

```
« 1 100 START NEXT »   →   Time: .387_s
```

TLVER: XLIB 803A40, 776 74

Object Types

PROLOG	(i)	Type	Object	Example
02933	1	0	Real number	1.2345
02977	2	1	Complex number	(2.3,4.5)
02A2C	3	2	String	"ABC"
029E8	4 <	3	Real array	[1 2 3]
		4	Complex array	[(1,2) (3,4)]
02A74	5	5	List	{ "ABC" Var }
02E48	6	6	Global name	X
02E6D	7	7	Local name	y
02D9D	8	8	Program	« A 2 + »
02AB8	9	9	Algebraic	'X=Y^2'
02A4E	B	10	Binary integer	# 247d
02B1E	C	11	Graphics object	Graphic 131 × 64
02AFC	D	12	Tagged object	Dist: 34.45
02ADA	E	13	Unit object	32_ft/s^2
02E92	0F	14	XLIB name	XLIB 766 1
02A96	2F	15	Directory	DIR ... END
02B40	8F	16	Library	Library 766: ...
02B62	9F	17	Backup object	Backup HOMEDIR
—		18	Built-in function	SIN
—		19	Built-in command	SWAP
02B88	AF	26	Library Data	Library Data

02911	1F	20	System Binary
02955	3F	21	long Real
0299D	4F	22	Long Complex
02ADA	5F	23	Linked Array
029BF	6F	24	Character
02DCC	7F	25	Code
		27	External

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

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. The command TRANSIO may be used to assert the current translation code.

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.

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

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).

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
−11, and −12	**Binary Integer Base**	**−11**	**−12**	DEC
	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	Rectangular	Clear	Clear	
and	Cylindrical Polar	Clear	Set	
-16	Spherical Polar	Set	Set	
Trigonometric Mode Flags		**-17**	**-18**	Degrees
-17	Degrees	Clear	Clear	
and	Radians	Set	Clear	
-18	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

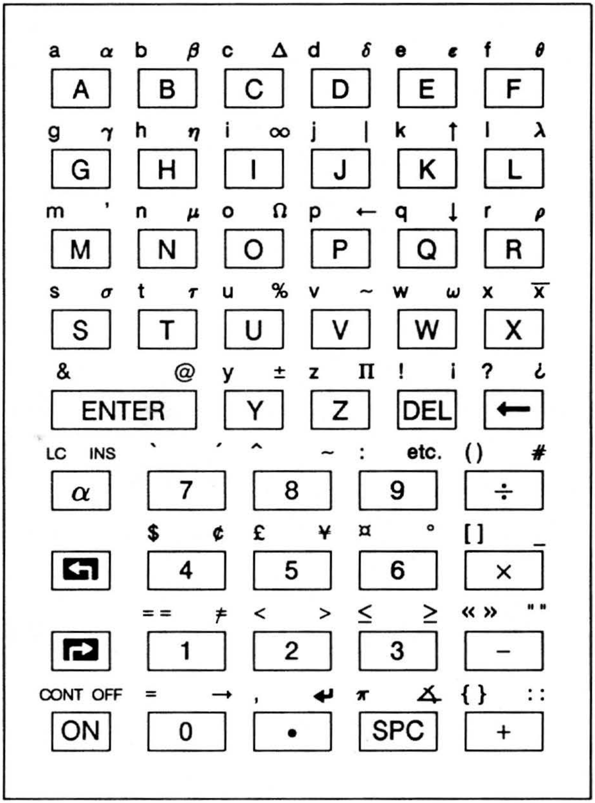
Flag	Description	Clear	Set	Default
I/O and Printing Flags				
-33	I/O Device	Serial	IR	Serial
-34	Printing Device	IR	Serial	IR
-35	I/O Data Format	ASCII	Binary	ASCII
-36	RECV Overwrite	New variable	Overwrite	New
-37	Double-Spaced Print	Single	Double	Single
-38	Linefeed	Inserts LF	Suppresses LF	Inserts
-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

Alpha Keyboard



The HP 48 Programmer's ToolKit