The HP 48
Programmer’s ToolKit

James Donnelly
The HP 48

Programmer’s ToolKit

James Donnelly
Acknowledgements

Special thanks go to Dan Allen, Scott Burke, Ray Depew, Alonzo Gariety, Joseph Horn, 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.
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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Lib Id</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSCAT</td>
<td>System Program</td>
<td></td>
<td>Character Set Catalog</td>
</tr>
<tr>
<td>LBLD</td>
<td>System Program</td>
<td></td>
<td>Menu Label Builder</td>
</tr>
<tr>
<td>FCLIB</td>
<td>Library</td>
<td>775</td>
<td>Flag Catalog</td>
</tr>
<tr>
<td>TLLIB</td>
<td>Library</td>
<td>776</td>
<td>Tool Library</td>
</tr>
<tr>
<td>TBLIB</td>
<td>Library</td>
<td>777</td>
<td>Title Browser</td>
</tr>
<tr>
<td>DBLIB</td>
<td>Library</td>
<td>778</td>
<td>Data Browser</td>
</tr>
</tbody>
</table>

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 \( \text{\textless} \) 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 \( \langle \text{DROP SWAP} \rangle \) 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.

Note: The libraries used by example programs must be installed before an example program is either entered or downloaded into the HP 48. For instance, the LUNCH example on page 21 uses commands from the Data Browser and the Tool Library. Therefore the Data Browser and Tool Library must be installed before LUNCH is entered, otherwise the library commands (such as DBR or PRVOB) will be interpreted as names, and the example programs will not work properly.
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  DCT  BIN
     41  065  101  01000001  5X9
     42  066  102  01000010  A
     43  067  103  01000011  A
     44  068  104  01000100  5X7
     45  069  105  01000101  A
     46  070  106  01000110  A
     47  071  107  01000111  A
     48  072  110  01001000  TIO 1
```

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  DCT  BIN
     BB  136  210  10001000  5X9
     BA  137  211  10001001  \n
     BC  140  214  10001100  5X7
     BD  141  215  10001101  \Ga
     BE  142  216  10001110  \n
     BF  143  217  10001111  \n```

The display below shows the display with character codes 136–143 displayed and TRANSIO set to 3:
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 $-\text{16}$ or $+\text{16}$ menu keys to move the pointer backwards or forwards 16 characters.
- Press the $-\text{32}$ or $+\text{32}$ menu keys to move the pointer backwards or forwards 32 characters. For \(A\rightarrow Z\), \(+\text{32}\) yields lowercase \(a\rightarrow z\).
- Press the $-\text{64}$ or $+\text{64}$ menu keys to move the pointer backwards or forwards 64 characters. For \(A\rightarrow Z\), $-\text{64}$ 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 \(A: 65:\)

<table>
<thead>
<tr>
<th>HOME (4: )</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROBE (3: )</td>
</tr>
<tr>
<td>HYP (2: )</td>
</tr>
<tr>
<td>MATR (1: )</td>
</tr>
<tr>
<td>VECTR</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>A: 65</td>
</tr>
</tbody>
</table>

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.
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 000000000000000101008282006444C0082820001010000000
"10Ω" "200Ω" "500Ω" )
}

GROB 21 8 00000000A00000A00000A000CFBF7000A00000A00000A000
"CAPACITOR"
}

GROB 21 8 0000000040000041000045000F7510004500004100004000
"GROUND"
}

GROB 21 8 000000000000000750002150E77750002450000720000000
"VCC"
}
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:

![Label Builder Grid]

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 x 8
1: Reg: Graphic 21 x 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:

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 x 5
1: Reg: Graphic 5 x 5
```
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:

This display shows 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:

<table>
<thead>
<tr>
<th>System Flag Catalog</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 C PRINCIPAL SOLUTION</td>
</tr>
<tr>
<td>2 C SYMBOLIC CONSTANTS</td>
</tr>
<tr>
<td>3 C NUMERICAL RESULTS</td>
</tr>
<tr>
<td>4 C NOT USED</td>
</tr>
<tr>
<td>5 S WORDSIZE 1</td>
</tr>
<tr>
<td>6 S WORDSIZE 2</td>
</tr>
<tr>
<td>7 S WORDSIZE 4</td>
</tr>
<tr>
<td>8 S WORDSIZE 8</td>
</tr>
</tbody>
</table>

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 `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)

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

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

Data Browser
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. The title string may contain 21 characters. Longer strings will be
truncated to 20 characters with a trailing ellipsis (…) character. 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.

- **/goto** 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 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:

<table>
<thead>
<tr>
<th>ADDRESS LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: JOHN DOE</td>
</tr>
<tr>
<td>Addr: 9876 WINCHESTER...</td>
</tr>
<tr>
<td>City: CORVALLIS</td>
</tr>
<tr>
<td>St.: OR</td>
</tr>
<tr>
<td>Zip: 97330</td>
</tr>
<tr>
<td>Ph#: 503-555-1212</td>
</tr>
</tbody>
</table>

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.

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:

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 \texttt{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 \texttt{edit\_flag} positive to check the user's input.

**Searching for Data.** Pressing $\alpha$ produces a prompt for a search string:

```
{ HOME }  PRG
Search for:
```

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 $\text{ATTN}$, or the search string can be accepted by pressing $\text{ENTER}$.

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.

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:

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

```
```
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:

```
Choose a planet:

<table>
<thead>
<tr>
<th>MERCURY</th>
<th>VENUS</th>
<th>EARTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>MARS</td>
<td>SATURN</td>
<td>JUPITER</td>
</tr>
<tr>
<td>URANUS</td>
<td>NEPTUNE</td>
<td>PLUTO</td>
</tr>
</tbody>
</table>

SUN  MOON TEMP DIST ORBIT QUIT
```

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

```
�数 Choose a number:

<table>
<thead>
<tr>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
</tbody>
</table>

QUIT
```

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: \{ \textit{current\_item} \textit{first\_row} \textit{title} \}
This list is similar to the level 1 input list. The real number \textit{current\_item} is the index in the data list of the underscored data item when the Title Browser terminated. The real number \textit{first\_row} specifies the first row of data elements that appeared in the display when the Title Browser terminated. The \textit{title} is the same as the input parameter.

Level 2: \textit{current\_item}
The real number \textit{current\_item} is the index in the data list of the underscored data item when the Title Browser terminated.

Level 1: \textit{terminator\_key}
The \textit{terminator\_key} indicates how the user terminated the Title Browser:

0: Zero is returned when the user presses \([\text{ATTN}]\).
1: One is returned when the user presses \([\text{ENTER}]\).
\(-n:\) If the result is a negative number, the absolute value indicates which menu key was pressed.
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.

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

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

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):

```plaintext
"MERCURY" "VENUS" "EARTH" "MARS" "SATURN"
"JUPITER" "URANUS" "NEPTUNE" "PLUTO"

"SUN" "MOON" "TEMP" "DIST" "ORBIT" "QUIT"
1 1 " Choose a planet: "
TBR
```
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](image)

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 ( ) PYVIEW
>
```

![Dotted Line](image)
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 \rightarrow LIST \emptyset VFIND \rightarrow 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 \rightarrow LIST \emptyset VFIND \rightarrow 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
____________________________________________________________________
Position 3
```
```
2 19 69 3
Position 2
```
```
(3,4) →
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.

A meta-object composed of three objects.

An object in level 1 and a meta-object beginning at level 2.

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 ... \}.

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 \}.

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.
- \( \rightarrow \) 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:

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

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTRACT</td>
<td>Returns the mth element from n lists</td>
</tr>
<tr>
<td>LSORT</td>
<td>Sorts a series of n lists based on the mth element</td>
</tr>
<tr>
<td>QSORT</td>
<td>Sorts a series of objects</td>
</tr>
<tr>
<td>VFIND</td>
<td>Finds variables in user memory</td>
</tr>
<tr>
<td>→WORDS</td>
<td>Separates a string into individual words</td>
</tr>
</tbody>
</table>
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)

```
EQN VARS DROP DTAG
OBJ
IF DUP THEN
  MZ2 DUP 1 SWAP
START
  MDT DUP VTYPE
  IF -1 SAME
  THEN
    MAH
  ELSE
    MAH2
  END
  NEXT MM→L
ELSE
  DROP
  ( ) ( )
END
"Undefined" →TAG SWAP
"Defined" →TAG
```

Tool Library
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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELCOL</td>
<td>Deletes a column from an array</td>
</tr>
<tr>
<td>DELROW</td>
<td>Deletes a row from an array</td>
</tr>
<tr>
<td>EXCOL</td>
<td>Exchanges two columns in an array</td>
</tr>
<tr>
<td>EXROW</td>
<td>Exchanges two rows in an array</td>
</tr>
<tr>
<td>GETCOL</td>
<td>Extracts a column from an array</td>
</tr>
<tr>
<td>GETROW</td>
<td>Extracts a row from an array</td>
</tr>
<tr>
<td>INSCOL</td>
<td>Inserts a column into an array</td>
</tr>
<tr>
<td>INSROW</td>
<td>Inserts a row into an array</td>
</tr>
<tr>
<td>PUTCOL</td>
<td>Replaces a column in an array</td>
</tr>
<tr>
<td>PUTROW</td>
<td>Replaces a row in an array</td>
</tr>
</tbody>
</table>

GRAPHICS

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PX+</td>
<td>Adds two graphics pixel coordinates</td>
</tr>
<tr>
<td>PX-</td>
<td>Subtracts two graphics pixel coordinates</td>
</tr>
<tr>
<td>PXOFF</td>
<td>Clears a pixel in an arbitrary graphics object</td>
</tr>
<tr>
<td>PXON</td>
<td>Sets a pixel in an arbitrary graphics object</td>
</tr>
<tr>
<td>PX?</td>
<td>Tests a pixel in an arbitrary graphics object</td>
</tr>
<tr>
<td>PX→R</td>
<td>Converts pixel coordinates into two real numbers</td>
</tr>
<tr>
<td>ROTATE</td>
<td>Rotates a graphics object</td>
</tr>
<tr>
<td>R→PX</td>
<td>Converts two real numbers into pixel coordinates</td>
</tr>
</tbody>
</table>

LIST MANIPULATION

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAR</td>
<td>Returns the first object of a list</td>
</tr>
<tr>
<td>CDR</td>
<td>Returns a list minus its first object</td>
</tr>
<tr>
<td>CUT</td>
<td>Splits a list into the first and remaining objects</td>
</tr>
<tr>
<td>EXTRACT</td>
<td>Returns the mth element from each of a series of lists</td>
</tr>
<tr>
<td>LSORT</td>
<td>Sorts a series of lists based on the mth element</td>
</tr>
<tr>
<td>NXTOB</td>
<td>Returns the next choice from a list of choices</td>
</tr>
<tr>
<td>PRVOB</td>
<td>Returns the previous choice from a list of choices</td>
</tr>
<tr>
<td>SPLIT</td>
<td>Splits a list into two lists</td>
</tr>
<tr>
<td>REPLACE</td>
<td>Replaces all occurrences of an object in a list</td>
</tr>
<tr>
<td>REVERSE</td>
<td>Reverses the order of objects in a list</td>
</tr>
<tr>
<td>ROTATE</td>
<td>Rotates the objects in a list</td>
</tr>
<tr>
<td>→SET</td>
<td>Removes duplicate objects from a list</td>
</tr>
</tbody>
</table>
### META–OBJECT UTILITIES

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

### SET UTILITIES

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADJOIN</td>
<td>Adds an object to a list if it is unique</td>
</tr>
<tr>
<td>DIFF</td>
<td>Returns the set difference of two lists</td>
</tr>
<tr>
<td>INTERSECT</td>
<td>Returns the set intersection between two lists</td>
</tr>
<tr>
<td>SDIFF</td>
<td>Returns the set symmetric difference of two lists</td>
</tr>
<tr>
<td>→SET</td>
<td>Removes duplicate objects from a list</td>
</tr>
<tr>
<td>UNION</td>
<td>Returns the set union of two lists</td>
</tr>
</tbody>
</table>

### STACK MANIPULATION

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEEP</td>
<td>Keeps the bottom $n$ objects on the stack</td>
</tr>
<tr>
<td>MREVERSE</td>
<td>Reverses the order of the first $n$ stack objects</td>
</tr>
<tr>
<td>NDUP</td>
<td>Creates $n$ copies of an object</td>
</tr>
<tr>
<td>QSORT</td>
<td>Sorts $n$ objects on the stack</td>
</tr>
<tr>
<td>SRLL</td>
<td>Rotates $n$ objects on the stack up $m$ times</td>
</tr>
<tr>
<td>SRLLD</td>
<td>Rotates $n$ objects on the stack down $m$ times</td>
</tr>
<tr>
<td>SXCH</td>
<td>Exchanges objects at levels $m$ and $n$</td>
</tr>
</tbody>
</table>
STRING MANIPULATION

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

OTHER COMMANDS

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOW</td>
<td>Returns the day of the week given a date</td>
</tr>
<tr>
<td>DOY</td>
<td>Returns the day of the year given a date</td>
</tr>
<tr>
<td>EQNVARS</td>
<td>Returns a list of global variables in an equation or program</td>
</tr>
<tr>
<td>VFIN</td>
<td>Find all occurrences of a variable or object type in user memory</td>
</tr>
<tr>
<td>XTIME</td>
<td>Calculates execution times</td>
</tr>
</tbody>
</table>

Error Messages

The Tool Library contains four new error messages:

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>30801</td>
<td>198657</td>
<td>Invalid Pos 1 Meta - Obj</td>
</tr>
<tr>
<td>30802</td>
<td>198658</td>
<td>Invalid Pos 2 Meta - Obj</td>
</tr>
<tr>
<td>30803</td>
<td>198659</td>
<td>Empty Meta - Obj</td>
</tr>
<tr>
<td>30804</td>
<td>198660</td>
<td>Inconsistent Data</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>Any object</td>
</tr>
<tr>
<td>x or y</td>
<td>Real number</td>
</tr>
<tr>
<td>(x,y)</td>
<td>Complex number</td>
</tr>
<tr>
<td>z</td>
<td>Real or complex number</td>
</tr>
<tr>
<td>m or n</td>
<td>Positive integer real number (rounded if non-integer)</td>
</tr>
<tr>
<td>#n or #m</td>
<td>Binary integer</td>
</tr>
<tr>
<td>&quot;string&quot;</td>
<td>Character string</td>
</tr>
<tr>
<td>&quot;chr&quot;</td>
<td>Character string containing only one character</td>
</tr>
<tr>
<td>{list}</td>
<td>List of objects</td>
</tr>
<tr>
<td>grob</td>
<td>Graphics object</td>
</tr>
<tr>
<td>{#x #y}</td>
<td>Pixel coordinates</td>
</tr>
<tr>
<td>date</td>
<td>Date in current date format</td>
</tr>
<tr>
<td>meta</td>
<td>Meta-object (see Meta-Objects)</td>
</tr>
<tr>
<td>type</td>
<td>Object type (see Object Types)</td>
</tr>
<tr>
<td>[vector]</td>
<td>Real or complex vector</td>
</tr>
<tr>
<td>[[matrix]]</td>
<td>Real or complex matrix</td>
</tr>
<tr>
<td>'global'</td>
<td>Global name</td>
</tr>
<tr>
<td>T/F</td>
<td>Test result: 0 (false) or non-zero (true)</td>
</tr>
</tbody>
</table>

Meta-object utilities are described with a notation presented in Meta-Objects.
ADJOIN
Adds an object to a list if the object is not a member of the list.

ADJOIN
{list₁} obj → {list₂}

Examples:

{ 11 22 33 } 33 → { 11 22 33 }

{ 11 22 33 } 44 → { 11 22 33 44 }

Related Commands: DIFF, INTERSECT, SDIFF, →SET, UNION
CAR
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          | " " → " "                          |
|             | "string₁" → "string₂"               |
|             | \{ \} → \{ \}                       |
|             | \{obj₁ ... objₙ\} → obj₁           |

Examples:

| "ABCD"      | → "A"                             |
| (3 9 8 2)    | → 3                               |

Related Commands: CDR, CUT, EXTRACT, NXTOB, PRVOB, SPLIT, REPLACE, REVERSE, ROTATE, →SET
The command CDR may be used to remove the first object from a list or the first character from a string.

<table>
<thead>
<tr>
<th>CDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>&quot;string₁&quot;</td>
</tr>
<tr>
<td>{ }</td>
</tr>
<tr>
<td>{obj₁ ... objₙ}</td>
</tr>
</tbody>
</table>

Examples:

"ABCD" → "BCD"

( 3982 ) → ( 982 )

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

\[
\begin{align*}
\text{[vector]}_1 & \quad n \quad \rightarrow \quad \text{[vector]}_2 \\
\text{[[matrix]}_1] & \quad n \quad \rightarrow \quad \text{[[matrix]}_2]
\end{align*}
\]

Examples:

\[
\begin{align*}
\begin{bmatrix}
3 & 9 & 8 & 2
\end{bmatrix} & \quad 3 \quad \rightarrow \quad \begin{bmatrix}
3 & 9 & 2
\end{bmatrix} \\
\begin{bmatrix}
11 & 22 & 33 \\
44 & 55 & 66
\end{bmatrix} & \quad 2 \quad \rightarrow \quad \begin{bmatrix}
11 & 33 \\
44 & 66
\end{bmatrix}
\end{align*}
\]

Related Commands: DELROW, EXCOL, EXROW, GETCOL, GETROW, INSCOL, INSROW, PUTCOL, PUTROW
DELROW
The command DELROW may be used to delete a row from a matrix. The matrix must have at least two rows.

\[
\text{DELROW} \quad [[\text{matrix}_1]] \ n \rightarrow [[\text{matrix}_2]]
\]

Example:

\[
\begin{bmatrix}
11 & 22 & 33 \\
44 & 55 & 66 \\
77 & 88 & 99
\end{bmatrix}
\rightarrow
\begin{bmatrix}
11 & 22 & 33 \\
77 & 88 & 99
\end{bmatrix}
\]

Related Commands: DELCOL, EXCOL, EXROW, GETCOL, GETROW, INSCOL, INSROW, PUTCOL, PUTROW
DIFF
Returns the set difference of two lists.

DIFF
\{list_a\} \{list_b\} → \{list_a AND NOT list_b\}

Examples:

\( \{1\,2\,3\,4\} \{5\,6\} \rightarrow \{1\,2\,3\,4\} \)

\( \{1\,2\,3\,4\} \{3\,4\,5\} \rightarrow \{1\,2\} \)

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

\[
date \rightarrow n
\]

Examples:

\[
5.181957 \rightarrow 138
\]

\[
3.231981 \rightarrow 82
\]

Related Command: DOW
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

<table>
<thead>
<tr>
<th>Input</th>
<th>Global: { names }</th>
<th>Local: { names }</th>
</tr>
</thead>
<tbody>
<tr>
<td>« program »</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'equation'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'name'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Examples:

'3*4' → Global: {} Local: {}

'R=\sqrt{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.
EXCOL
Exchanges two columns in an array.

EXCOL

\[
\begin{align*}
\text{[vector}_1\text{]} & \text{ col}_1 & \text{ col}_2 & \rightarrow & \text{[vector}_2\text{]} \\
\text{[[matrix}_1\text{]]} & \text{ col}_1 & \text{ col}_2 & \rightarrow & \text{[matrix}_2\text{]} \\
\end{align*}
\]

Examples:

\[
\begin{bmatrix}
1 & 2 & 3 & 4 \\
\end{bmatrix}
\begin{bmatrix}
2 & 3 \\
\end{bmatrix}
\rightarrow
\begin{bmatrix}
1 & 3 & 2 & 4 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
\end{bmatrix}
\begin{bmatrix}
1 & 2 \\
\end{bmatrix}
\rightarrow
\begin{bmatrix}
2 & 1 \\
4 & 3 \\
\end{bmatrix}
\]

Related Commands: DELCOL, DELROW, EXROW, GETCOL, GETROW, INSCOL, INSROW, PUTCOL, PUTROW
EXROW
Exchanges two rows in an array.

\[
\text{EXROW} \quad \begin{bmatrix} \text{[matrix}_1\text{]} \end{bmatrix} \text{ row}_1 \text{ row}_2 \rightarrow \begin{bmatrix} \text{[matrix}_2\text{]} \end{bmatrix}
\]

Example:

\[
\begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} 3 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 2 \end{bmatrix}
\]

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

\begin{verbatim}
EXTRACT
  {list$_1$} ... {list$_n$}  n  obj-number  →  obj$_1$ ... obj$_n$  n
\end{verbatim}

Example:

\begin{verbatim}
( 3 91 ) ( 1 78 ) ( 8 12 ) 3 2  →  91 78 12 3
\end{verbatim}

Related Commands: LSORT, MREVERSE
GETCOL
Returns a column from an array as a matrix consisting of 1-element rows.

<table>
<thead>
<tr>
<th>GETCOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>[vector] col → [[column data]]</td>
</tr>
<tr>
<td>[[matrix]] col → [[column data]]</td>
</tr>
</tbody>
</table>

Examples:

\[
\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} 2 \end{bmatrix}
\]

\[
\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 2 \\ 5 \end{bmatrix}
\]

Related Commands: DELCOL, DELROW, EXCOL, EXROW, GETROW, INSCOL, INSROW, PUTCOL, PUTROW
GETROW
Returns a row from an array as a vector.

<table>
<thead>
<tr>
<th>GETROW</th>
</tr>
</thead>
<tbody>
<tr>
<td>[vector] row → [row data]</td>
</tr>
<tr>
<td>[[matrix]] row → [row data]</td>
</tr>
</tbody>
</table>

Examples:

\[
\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \quad 1 \quad \rightarrow \quad \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}
\]

\[
\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad 2 \quad \rightarrow \quad \begin{bmatrix} 4 & 5 & 6 \end{bmatrix}
\]

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

<table>
<thead>
<tr>
<th>Lowercase</th>
<th>Uppercase</th>
</tr>
</thead>
<tbody>
<tr>
<td>61h–7Ah</td>
<td>41h–5Ah</td>
</tr>
<tr>
<td>E0h–F6h</td>
<td>C0h–D6h</td>
</tr>
<tr>
<td>F8h–FEh</td>
<td>D8h–DEh</td>
</tr>
</tbody>
</table>

Examples:

"JOHN SMITH"  →  "John Smith"

"sample sentence"  →  "Sample Sentence"

Related Commands: LCASE, UCASE
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.

\[
\text{INSCOL} \quad [\text{vector}_1] \ n \rightarrow \ [\text{vector}_2] \\
[[\text{matrix}_1]] \ n \rightarrow \ [[\text{matrix}_2]]
\]

Examples:

\[
\begin{bmatrix} 3 & 9 & 8 & 2 \end{bmatrix} \ 3 \rightarrow \ \begin{bmatrix} 3 & 9 & 0 & 8 & 2 \end{bmatrix}
\]

\[
\begin{bmatrix} (9,4) & (8,3) \end{bmatrix} \ 3 \rightarrow \ \begin{bmatrix} (9,4) & (8,3) & (0,0) \end{bmatrix}
\]

\[
\begin{bmatrix} 11 & 22 & 33 \ & 44 & 55 & 66 \ & 77 & 88 & 99 \end{bmatrix} \ 2 \rightarrow \ \begin{bmatrix} 11 & 0 & 22 & 33 \ & 44 & 0 & 55 & 66 \ & 77 & 0 & 88 & 99 \end{bmatrix}
\]

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

<table>
<thead>
<tr>
<th>INSROW</th>
</tr>
</thead>
<tbody>
<tr>
<td>[vector] n → [[matrix]]</td>
</tr>
<tr>
<td>[[matrix₁]] n → [[matrix₂]]</td>
</tr>
</tbody>
</table>

Examples:

\[
\begin{bmatrix}
3 & 9 & 8 & 2
\end{bmatrix}
1 \rightarrow
\begin{bmatrix}
0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
11 & 22 & 33 \\
44 & 55 & 66 \\
77 & 88 & 99
\end{bmatrix}
4 \rightarrow
\begin{bmatrix}
44 & 55 & 66 \\
77 & 88 & 99 \\
0 & 0 & 0
\end{bmatrix}
\]

**Related Commands:** DELCOL, DELROW, EXCOL, EXROW, GETCOL, GETROW, INSCOL, PUTCOL, PUTROW
INTERSECT
Returns the set intersection between two lists.

\[
\text{INTERSECT} \quad \{\text{list}_a\} \quad \{\text{list}_b\} \quad \rightarrow \quad \{\text{list}_a \text{ AND } \text{list}_b\}
\]

Examples:

\[
(1 \ 2 \ 3 \ 4) \ (5 \ 6) \quad \rightarrow \quad ()
\]

\[
(1 \ 2 \ 3 \ 4) \ (3 \ 4 \ 5) \quad \rightarrow \quad (3 \ 4)
\]

Related Commands: ADJOIN, DIFF, SDIFF, \rightarrow \text{SET}, \text{UNION}
**KEEP**

Keeps the bottom \( n \) objects on the stack while deleting all objects above \( n \).

\[
\text{KEEP} \\
\ldots \text{obj}_n \ldots \text{obj}_1 \ n \rightarrow \text{obj}_n \ldots \text{obj}_1
\]

Example:

"AA" 32 7.1 "B" 2 \( \rightarrow \) 7.1 "B"

Related Commands: NDUP, SRL, SRLD, SXCH
LCASE
Converts each character in a string to lowercase.

```
LCASE
"string₁" → "string₂"
```

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

<table>
<thead>
<tr>
<th>Lowercase</th>
<th>Uppercase</th>
</tr>
</thead>
<tbody>
<tr>
<td>61h–7Ah</td>
<td>41h–5Ah</td>
</tr>
<tr>
<td>E0h–F6h</td>
<td>C0h–D6h</td>
</tr>
<tr>
<td>F8h–FEh</td>
<td>D8h–DEh</td>
</tr>
</tbody>
</table>

Example:

"SAMPLE SENTENCE" → "sample sentence"

Related Commands: ICAPS, UCASE
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).

\[
\text{LSORT} \quad \{\text{list}_1\} \ldots \{\text{list}_n\} \quad n \quad m \rightarrow \{\text{list}_1\} \ldots \{\text{list}_n\} \quad n
\]

Example:

\[
\begin{align*}
(39) & \ (17) \ (82) \ 3 \ 1 \rightarrow \\
(17) & \ (39) \ (82) \ 3
\end{align*}
\]

Related Commands: EXTRACT, MREVERSE, QSORT
LTRIM
Removes leading space and tab (#09h) characters from a string.

LTRIM
"string₁" → "string₂"

Example:
" SAMPLE STRING " → "SAMPLE STRING "

Related Commands: RTRIM, TRIM, →WORDS
MAH
Adds an object to the head of a meta-object.

<table>
<thead>
<tr>
<th>MAH</th>
</tr>
</thead>
<tbody>
<tr>
<td>meta₁</td>
</tr>
<tr>
<td>&lt; ... &gt;</td>
</tr>
</tbody>
</table>

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
MAH2
Adds an object to the head of a meta-object in position 2.

<table>
<thead>
<tr>
<th>MAH2</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \text{meta}_2 ; \text{meta}_1 ; \text{obj} \rightarrow \text{meta}_2' ; \text{meta}_1 ]</td>
</tr>
<tr>
<td>[ &lt; \text{meta}_2 &gt; ; &lt; \text{meta}_1 &gt; ; \text{obj} \rightarrow ; &lt; \text{obj} ; \text{meta}_2 &gt; ; &lt; \text{meta}_1 &gt; ]</td>
</tr>
</tbody>
</table>

Example:

\[ 21 \; 32 \; 2 \; 2.3 \; 4.7 \; 2 \; 99 \rightarrow \; 99 \; 21 \; 32 \; 3 \; 2.3 \; 4.7 \; 2 \]

Related Commands: MAH, MAM2, MAT, MAT2, MDH, MDH2, MDT, MDT2, ML→M, MM→L, MREVERSE, MSWAP, MZ2
MAM2
Concatenates two meta-objects.

\[
\text{MAM2} \quad \text{meta}_2 \text{ meta}_1 \text{ obj} \rightarrow \text{meta}_{2+1}
\]

\[
< \text{meta}_2 > < \text{meta}_1 > \rightarrow < \text{meta}_{2+1} >
\]

Example:

\[
21 \ 32 \ 47 \ 3 \ 7.3 \ 4.8 \ 2 \rightarrow 21 \ 32 \ 47 \ 7.3 \ 4.8 \ 5
\]

Related Commands: MAH, MAH2, MAT, MAT2, MDH, MDH2, MDT, MDT2, ML→M, MM→L, MREVERSE, MSWAP, MZ2
MAT
Adds an object to the tail of a meta-object.

MAT

\[
\begin{align*}
\text{meta}_1 \quad \text{obj} & \rightarrow \quad \text{meta}_2 \\
< \ldots > \quad \text{obj} & \rightarrow \quad < \ldots \ \text{obj} >
\end{align*}
\]

Example:

\[
21 \ 32 \ 47 \ 3 \ 99 \ \rightarrow \ 21 \ 32 \ 47 \ 99 \ 4
\]

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

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

\[
\begin{align*}
\text{MAT2} & : \quad \text{meta}_2 \quad \text{meta}_1 \quad \text{obj} \quad \rightarrow \quad \text{meta}_2' \quad \text{meta}_1 \\
< \text{meta}_2 > \quad < \text{meta}_1 > \quad \text{obj} \quad \rightarrow \quad < \text{meta}_2 \quad \text{obj} > \quad < \text{meta}_1 >
\end{align*}
\]

Example:

\[
21 \quad 32 \quad 2 \quad 2.3 \quad 4.7 \quad 2 \quad 99 \quad \rightarrow \quad 21 \quad 32 \quad 99 \quad 3 \quad 2.3 \quad 4.7 \quad 2
\]

**Related Commands:** MAH, MAH2, MAM2, MAT, MDH, MDH2, MDT, MDT2, ML→M, MM→L, MREVERSE, MSWAP, MZ2
MDH
Extracts an object from the head of a meta-object.

<table>
<thead>
<tr>
<th>MDH</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{meta}_1 \rightarrow \text{meta}_2 \ obj$</td>
</tr>
<tr>
<td>$&lt; \ obj \ ... \ &gt; \rightarrow &lt; \ ... \ &gt; \ obj$</td>
</tr>
</tbody>
</table>

Example:

$\begin{align*}
99 & \ 21 \ 32 \ 47 \ 4 \\
\rightarrow & \ 21 \ 32 \ 47 \ 3 \ 99
\end{align*}$

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

\[
99 \; 21 \; 32 \; 3 \; 2.3 \; 4.7 \; 2 \rightarrow 21 \; 32 \; 2 \; 2.3 \; 4.7 \; 2 \; 99
\]

Related Commands: MAH, MAH2, MAM2, MAT, MAT2, MDH, MDT, MDT2, ML→M, MM→L, MREVERSE, MSWAP, MZ2
MDT
Extracts an object from the tail of a meta-object.

\[
\text{MDT} \quad \text{meta}_1 \rightarrow \text{meta}_2 \text{ obj}
\]

\[
< \ldots \text{ obj } > \rightarrow < \ldots > \text{ obj}
\]

Example:

\[
21 \ 32 \ 47 \ 99 \ 4 \rightarrow 21 \ 32 \ 47 \ 3 \ 99
\]

Related Commands: MAH, MAH2, MAM2, MAT, MAT2, MDH, MDH2, MDT2, ML→M, MM→L, MREVERSE, MSWAP, MZ2
MDT2
Extracts an object from the tail of a meta-object in position 2.

\[
\text{MDT2} \\
\text{meta}_2 \text{ meta}_1 \rightarrow \text{meta}_2' \text{ meta}_1 \text{ obj} \\
< \text{meta}_2 \text{ obj} > < \text{meta}_1 > \rightarrow < \text{meta}_2' > < \text{meta}_1 > \text{ obj}
\]

Example:

\[
21 \ 32 \ 99 \ 3 \ 2.3 \ 4.7 \ 2 \rightarrow 21 \ 32 \ 2 \ 2.3 \ 4.7 \ 2 \ 99
\]

Related Commands: MAH, MAH2, MAM2, MAT, MAT2, MDH, MDH2, MDT, ML→M, MM→L, MREVERSE, MSWAP, MZ2
ML→M
Converts two lists into meta-objects.

\[
\begin{align*}
\{\text{list}_2\} \quad \{\text{list}_1\} & \rightarrow \quad \text{meta}_2 \quad \text{meta}_1 \\
\{\text{list}_2\} \quad \{\text{list}_1\} & \rightarrow \quad <\text{meta}_2> \quad <\text{meta}_1>
\end{align*}
\]

Example:

\[
\{11 \quad 22\} \quad \{3.1 \quad 4.2 \quad 5.1\} \rightarrow \\
11 \quad 22 \quad 2 \quad 3.1 \quad 4.2 \quad 5.1 \quad 3
\]

Related Commands: MAH, MAH2, MAM2, MAT, MAT2, MDH, MDH2, MDT, MDT2, MM→L, MREVERSE, MSWAP, MZ2
**MM→L**
Converts two meta-objects into lists.

\[
\text{MM→L} \\
\begin{align*}
\text{meta}_2 & \quad \text{meta}_1 \quad \rightarrow \quad \{\text{list}_2\} \quad \{\text{list}_1\} \\
< \text{meta}_2 > & \quad < \text{meta}_1 > \quad \rightarrow \quad \{\text{list}_2\} \quad \{\text{list}_1\}
\end{align*}
\]

Example:

\[
11 \quad 22 \quad 2 \quad 3.1 \quad 4.2 \quad 5.1 \quad 3 \quad \rightarrow \\
\quad \{11 \quad 22\} \quad \{3.1 \quad 4.2 \quad 5.1\}
\]

**Related Commands:** MAH, MAH2, MAM2, MAT, MAT2, MDH, MDH2, MDT, MDT2, ML→M, MREVERSE, MSWAP, MZ2
MREVERSE
Reverses the order of \( n \) objects on the stack. This command will reverse the order of 5000 stack items about two seconds.

\[
\begin{align*}
\text{MREVERSE} & \quad \text{obj}_1 \ldots \text{obj}_n \ n \rightarrow \ \text{obj}_n \ldots \text{obj}_1 \ n \\
& \quad \text{meta}_1 \rightarrow \ \text{meta}_2 \\
& \quad < \text{obj}_1 \ \text{obj}_2 \ \text{obj}_3 > \rightarrow \ < \text{obj}_3 \ \text{obj}_2 \ \text{obj}_1 >
\end{align*}
\]

Example:

\[
11 \ 22 \ .2 \ 3.1 \ 2 \ 5 \ \rightarrow \ 2 \ 3.1 \ .2 \ 22 \ 11 \ 5
\]

Related Commands: KEEP, MAH, MAH2, MAM2, MAT, MAT2, MDH, MDH2, MDT, MDT2, ML→M, MM→L, MSWAP, MZ2, NDUP, SRL, SRLD, SXCH
MSWAP
Swaps two meta-objects on the stack.

\[
\begin{align*}
\text{meta}_2 & \quad \text{meta}_1 & \rightarrow & \quad \text{meta}_1 & \quad \text{meta}_2 \\
< \text{meta}_2 > & \quad < \text{meta}_1 > & \rightarrow & \quad < \text{meta}_1 > & \quad < \text{meta}_2 >
\end{align*}
\]

Example:

\[
11 \quad 22 \quad 2 \quad 3.1 \quad 4.2 \quad 5.1 \quad 3 \quad \rightarrow \quad 3.1 \quad 4.2 \quad 5.1 \quad 3 \quad 11 \quad 22 \quad 2
\]

Related Commands: MAH, MAH2, MAM2, MAT, MAT2, MDH, MDH2, MDT, MDT2, ML→M, MM→L, MREVERSE, MZ2
MZ2
Places an empty meta-object in meta-stack position 2.

\[
\begin{align*}
\text{meta}_1 & \rightarrow \text{meta}_{\text{empty}} \text{ meta}_1 \\
< \text{meta}_1 > & \rightarrow < > < \text{meta}_1 >
\end{align*}
\]

Example:

\[
3.1 \ 4.2 \ 5.1 \ 3 \ \rightarrow \ 0 \ 3.1 \ 4.2 \ 5.1 \ 3
\]

Related Commands: MAH, MAH2, MAM2, MAT, MAT2, MDH, MDH2, MDT, MDT2, ML→M, MM→L, MREVERSE, MSWAP
NDUP
Creates \( n \) copies of an object on the stack. If \( n \) is zero, no objects will be returned.

\[
\text{NDUP} \quad \text{obj} \ n \rightarrow \text{obj} \ldots \text{obj}
\]

Examples:

\[
\begin{align*}
23 \ 0 & \rightarrow \\
5.1 \ 3 & \rightarrow \ 5.1 \ 5.1 \ 5.1
\end{align*}
\]

Related Commands: KEEP, MREVERSE, SRL, SRLLD, SXCH
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.

\[
\{\text{obj}_1 \ldots \text{obj}_n\} \quad \text{obj}_m \rightarrow \text{obj}_{m+1}
\]

Examples:

\[
(\quad) \quad \text{"FRED"} \quad \rightarrow \quad \text{"FRED"}
\]
\[
(11 \quad 22 \quad 33) \quad 22 \quad \rightarrow \quad 33
\]
\[
(11 \quad 22 \quad 33) \quad 33 \quad \rightarrow \quad 11
\]

Related Commands: CDR, CUT, EXTRACT, LSORT, PRVOB, SPLIT, REPLACE, REVERSE, ROTATE, \rightarrow\text{SET}
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.

\[
\text{PRVOB} \quad \{\text{obj}_1 \ldots \text{obj}_n\} \quad \text{obj}_m \rightarrow \text{obj}_{m-1}
\]

Examples:

\[
\{ \} \quad \text{"FRED"} \quad \rightarrow \quad \text{"FRED"}
\]

\[
\{ 11 \ 22 \ 33 \} \quad 22 \quad \rightarrow \quad 11
\]

\[
\{ 11 \ 22 \ 33 \} \quad 11 \quad \rightarrow \quad 33
\]

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

\[
\begin{align*}
& \text{PUTCHR} \\
& \quad \text{"string}_1\text{," position code } \rightarrow \text{ "string}_2\text{,"
} \\
& \quad \text{"string}_1\text{," position "string}_2\text{," } \rightarrow \text{ "string}_3\text{,"
}\end{align*}
\]

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
PUTCOL
Replaces a column of data in an array.

PUTCOL

\[ [[\text{matrix}_1]] \ col \ [\text{[new - col]}] \rightarrow \ [[\text{matrix}_2]] \]

Examples:

\[
\begin{bmatrix}
1 & 2 & 3 \\
\end{bmatrix}
\rightarrow
\begin{bmatrix}
1 & 44 & 3
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6
\end{bmatrix}
\rightarrow
\begin{bmatrix}
1 & 11 & 3 \\
4 & 22 & 6
\end{bmatrix}
\]

Related Commands: DELCOL, DELROW, EXCOL, EXROW, GETCOL, GETROW, INSCOL, INSROW, PUTROW
PUTROW
Replaces a row of data in an array.

PUTROW

\[
\begin{array}{c}
\text{[vector]} \ 	ext{row} \ 	ext{[new-row]} \rightarrow \ 	ext{[vector]} \\
\text{[[matrix]]} \ 	ext{row} \ 	ext{[new-row]} \rightarrow \ 	ext{[[matrix]]}
\end{array}
\]

Examples:

\[
\begin{align*}
\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} & 1 \begin{bmatrix} 4 & 5 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 4 & 5 & 6 \end{bmatrix} \\
\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 4 & 5 & 6 \end{bmatrix} & 2 \begin{bmatrix} 7 & 8 & 9 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 7 & 8 & 9 \end{bmatrix}
\end{align*}
\]

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

Adds two graphics pixel coordinates.

\[
\{ #_1 \, #_2 \} \quad \{ #_3 \, #_4 \} \rightarrow \quad \{ #_{1+3} \, #_{2+4} \}
\]

Example:

\[
( \#3d \, \#7d \) \quad ( \#6d \, \#1d \) \rightarrow \quad ( \#9d \, \#8d \)
\]

Related Command: PX−
PX-
Subtracts two graphics pixel coordinates.

\[
\text{PX-} \quad \{ #_1, #_2 \} \{ #_3, #_4 \} \to \{ #_{1-3}, #_{2-4} \}
\]

Example:

\[
\langle \#23d \#54d \rangle \langle \#6d \#1d \rangle \to \langle \#17d \#53d \rangle
\]

Related Command: PX+
PXOFF
Clears a pixel in an arbitrary graphics object.

<table>
<thead>
<tr>
<th>PXOFF</th>
</tr>
</thead>
</table>
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?
**PXON**
Sets a pixel in an arbitrary graphics object.

```
PXON
  \text{grob} \{ \#_x \#_y \} \rightarrow \text{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 0300  \{ \#6d \#1d \} \rightarrow GROB 8 2 0304
```

**Related Commands:** PXOFF, PX?
PX?
Tests a pixel in an arbitrary graphics object.

PX?
\[
grob \{ #x \ #y \} \rightarrow \ T/F
\]

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:

\[
\begin{align*}
\text{GROB} & \ 8 \ 2 \ 0300 \ \{ \ #6d \ #1d \ \} \ \rightarrow \ 0 \\
\text{GROB} & \ 8 \ 2 \ 0304 \ \{ \ #6d \ #1d \ \} \ \rightarrow \ 1
\end{align*}
\]

Related Commands: PXOFF, PXON
**PX→R**
Converts a list of two binary integers to two real numbers.

\[
\text{PX→R} \quad \{ \ #\text{col} \ #\text{row} \} \quad \rightarrow \quad \text{col} \ \text{row}
\]

Example:

\[
\{ \ #4\text{d} \ #18\text{d} \ } \quad \rightarrow \quad 4 \ 18
\]

**Related Command:** R→PX
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).

<table>
<thead>
<tr>
<th>QSORT</th>
<th>( obj_1 \ldots obj_n \ n \rightarrow \ obj_? \ldots obj_? \ n )</th>
</tr>
</thead>
</table>

Examples:

\[
3 2 8 7 4 \rightarrow 2 3 7 8 4
\]

"FRED" "ANNE" "ZOE" 3 \( \rightarrow \) "ANNE" "FRED" "ZOE" 3

Related Commands: LSORT, MREVERSE
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**

<table>
<thead>
<tr>
<th>string₁</th>
<th>string_search</th>
<th>string repl →</th>
<th>string₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>list₁</td>
<td>obj_search</td>
<td>obj repl →</td>
<td>list₂</td>
</tr>
</tbody>
</table>

**Examples:**

```
"JOHN" "H" "A" → "JOAN"

"ABCD" "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
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 repacking 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.

Examples:

"ABCD" → "DCBA"

{ 1 2 3 4 5 } → { 5 4 3 2 1 }

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

<table>
<thead>
<tr>
<th>$x&lt;0$</th>
<th>Rotates left</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x=0$</td>
<td>No change</td>
</tr>
<tr>
<td>$x&gt;0$</td>
<td>Rotates right</td>
</tr>
</tbody>
</table>

Graphics objects are rotated $90^\circ$ to the left for $x<0$, or $90^\circ$ 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 \(\rightarrow\) GROB -1 ROTATE PICT STO {} PVIEW 
```

```
"123" 2 \(\rightarrow\) 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, \(\rightarrow\)SET
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.

<table>
<thead>
<tr>
<th>RPTSTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;string&quot; $n$ → &quot;string ... string&quot;</td>
</tr>
</tbody>
</table>

Examples:

"ABC" 0 → ""

"ABC" 3 → "ABCABCABC"

Related Command: STRCON
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
R→PX
Converts two real numbers to a list of two binary integers.

\[
\begin{array}{c}
\text{(col \ row)} \\
\rightarrow \\
\{ \#\text{col} \ #\text{row} \}
\end{array}
\]

Example:

\[
\begin{align*}
45 & \quad 37 \\
\rightarrow & \quad \{ \#45d \ #37d \}
\end{align*}
\]

Related Command: PX→R
SDIFF
Returns the set symmetric difference of two lists.

\[
\text{SDIFF} \quad \{\text{list}_a\} \quad \{\text{list}_b\} \quad \rightarrow \quad \{\text{list}_a \xor \text{list}_b\}
\]

Examples:

\[
\begin{align*}
\{1 \ 2 \ 3 \ 4\} \ {5 \ 6} & \quad \rightarrow \quad \{1 \ 2 \ 3 \ 4 \ 5 \ 6\} \\
\{1 \ 2 \ 3 \ 4\} \ {3 \ 4 \ 5} & \quad \rightarrow \quad \{1 \ 2 \ 5\}
\end{align*}
\]

Related Commands: ADJOIN, DIFF, INTERSECT, →SET, UNION
→SET
Removes duplicate objects from a list.

\[ \{ \text{list}_1 \} \rightarrow \{ \text{list}_2 \} \]

Examples:

\[ \{ 1 \ 2 \ 3 \ 4 \} \rightarrow \{ 1 \ 2 \ 3 \ 4 \} \]

\[ \{ 4 \ 1 \ 2 \ 3 \ 2 \ 1 \} \rightarrow \{ 4 \ 1 \ 2 \ 3 \} \]

Related Commands: ADJOIN, INTERSECT, CAR, CDR, CUT, DIFF, EXTRACT, NXTOB, PRVOB, SDIFF, SPLIT, REPLACE, REVERSE, ROTATE, UNION
The command SPLIT may be used to divide a list or string into first $m$ and remaining components.

### SPLIT

- **""** $m$ $\rightarrow$ **""""**
- "string$_1$" $m$ $\rightarrow$ "string$_2$" "string$_3$"
- `{ } m$ $\rightarrow$ `{ } `{ }
- `{obj$_1$ ... obj$_n$} m$ $\rightarrow$ `{obj$_{m+1}$ ... obj$_n$} `{obj$_1$ ... obj$_m$}

### Examples:

- "ABCDE" 0 $\rightarrow$ "ABCDE" ""
- "ABCDE" 3 $\rightarrow$ "DE" "ABC"
- `{ } 3 $\rightarrow$ `{ } `{ }
- `{3 9 8 2 7} 0 $\rightarrow$ `{3 9 8 2 7} `{ }
- `{3 9 8 2 7} 2 $\rightarrow$ `{8 2 7} `{3 9 }

### Related Commands: CAR, CDR, CUT, REVERSE, ROTATE
SRLL
Rotates \( n \) objects on the stack up \( m \) times.

\[
\text{SRLL} \quad \text{obj}_1 \ldots \text{obj}_n \ n \ m \rightarrow \text{obj}_{n-m+1} \ldots \text{obj}_n \ \text{obj}_1 \ldots \text{obj}_{m+1}
\]

Example:

\[
11 \ 22 \ 33 \ 44 \ 55 \ 5 \ 2 \quad \rightarrow \quad 33 \ 44 \ 55 \ 11 \ 22
\]

Related Commands: KEEP, MREVERSE, NDUP, SRLLD, SXCH
SRLLD
Rotates $n$ objects on the stack down $m$ times.

\[
\text{SRLL} \quad \text{obj}_1 \ldots \text{obj}_n \ n \ m \rightarrow \text{obj}_{m+1} \ldots \text{obj}_n \ \text{obj}_1 \ldots \text{obj}_m
\]

Example:

\[
11 \ 22 \ 33 \ 44 \ 55 \ 5 \ 2 \rightarrow \ 44 \ 55 \ 11 \ 22 \ 33
\]

Related Commands: KEEP, MREVERSE, NDUP, SRL, SXCH
→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"
STRCON
Creates a string consisting of \( n \) repetitions of a character \textit{code}. Strings are created at a rate nearing 20,000 characters per second.

\[
\text{code} \ n \rightarrow \ "\text{string}" \\
\]

Examples:

\[
65 \ 0 \rightarrow \ "\" \\
65 \ 10 \rightarrow \ "AAAAAAAAAAA" \\
\]

Related Commands: PUTCHR, RPTSTR, SUBNUM
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 "
```

Note: If the string is longer than the specified number of spaces, the string will be truncated and an ellipsis character (...) will be added at the end.

Related Command: TRIM
SUBNUM
Returns the character code of the $n$th character of a string.

<table>
<thead>
<tr>
<th>SUBNUM</th>
<th>&quot;string&quot; n → code</th>
</tr>
</thead>
</table>

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
**SXCH**
Exchanges objects at levels \( m \) and \( n \) on the stack.

<table>
<thead>
<tr>
<th>SXCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \ldots \ obj_m \ldots \ obj_n \ldots \ m \ n \rightarrow \ldots \ obj_n \ldots \ obj_m \ldots )</td>
</tr>
</tbody>
</table>

**Example:**

\[
58 \ 22 \ 87 \ 34 \ 14 \ 4 \ 2 \ \rightarrow \ \ 58 \ 34 \ 87 \ 22 \ 14
\]

**Related Commands:** KEEP, MREVERSE, NDUP, QSORT, SRL, SRLLD
→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
"string₁" → "string₂"
```

Example:

"« → x « x SIN x / » »" →
"\langle \langle \rightarrow r \langle \langle x SIN x / \rangle \rangle \rangle "

Related Command: TIO→
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).

\[
\text{TIO→} \quad \text{"string}_1\text{"} \rightarrow \text{"string}_2\text{"}
\]

Example:

"\langle\langle \rightarrow r \langle\langle x \ SIN \ x / \rangle\rangle \rangle\rangle" \rightarrow
\quad \text{"} « \rightarrow x \langle\langle \rangle\rangle \langle\langle x \ SIN \ x / \rangle\rangle \rangle \rangle\rangle\text{"}

Related Command: →TIO
TRIM
Removes leading and trailing space and tab (#09h) characters from a string.

\[
\text{TRIM} \quad "\text{string}_1" \rightarrow "\text{string}_2"
\]

Example:

" SAMPLE STRING " \rightarrow "SAMPLE STRING"

Related Commands: LTRIM, RTRIM, STRCTR, \rightarrow WORDS
UCASE
Converts each character in a string to uppercase.

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

<table>
<thead>
<tr>
<th>Lowercase</th>
<th>Uppercase</th>
</tr>
</thead>
<tbody>
<tr>
<td>61h–7Ah</td>
<td>41h–5Ah</td>
</tr>
<tr>
<td>E0h–F6h</td>
<td>C0h–D6h</td>
</tr>
<tr>
<td>F8h–FEh</td>
<td>D8h–DEh</td>
</tr>
</tbody>
</table>

Example:

"sample sentence"  →  "SAMPLE SENTENCE"

Related Commands: ICAPS, LCASE
UNION
Returns the set union of two lists.

\[
\text{UNION} \quad \{\text{list}_a\} \quad \{\text{list}_b\} \quad \rightarrow \quad \{\text{list}_a \cup \text{list}_b\}
\]

Examples:

\[
(1 \ 2 \ 3 \ 4) \ (5 \ 6) \quad \rightarrow \quad (1 \ 2 \ 3 \ 4 \ 5 \ 6)
\]

\[
(1 \ 2 \ 3 \ 4) \ (3 \ 4 \ 5) \quad \rightarrow \quad (1 \ 2 \ 3 \ 4 \ 5)
\]

Related Commands: ADJOIN, DIFF, INTERSECT, SDIFF, \rightarrow\text{SET}
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

\[
\begin{align*}
\text{name} & \rightarrow \{\text{path}_1\} \ldots \{\text{path}_m\} n \\
\text{type} & \rightarrow \{\text{path}_1\} \ldots \{\text{path}_m\} n
\end{align*}
\]

Examples:

\[
\begin{align*}
4 & \rightarrow 0 \\
0 & \rightarrow (\text{HOME } X ) (\text{HOME REALS } Y ) 2 \\
'X' & \rightarrow (\text{HOME } X ) (\text{HOME REALS } X ) 2
\end{align*}
\]
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.

```
"word_1 ... word_n"  →  "word_1" ... "word_n"  n
```

Examples:

```
""  →  0

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

Related Commands: LTRIM, RTRIM, TRIM
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.

<table>
<thead>
<tr>
<th>XTIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>object → seconds</td>
</tr>
</tbody>
</table>

Example:

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Object</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Real number</td>
<td>1.2345</td>
</tr>
<tr>
<td>1</td>
<td>Complex number</td>
<td>(2.3, 4.5)</td>
</tr>
<tr>
<td>2</td>
<td>String</td>
<td>&quot;ABC&quot;</td>
</tr>
<tr>
<td>3</td>
<td>Real array</td>
<td>[1, 2, 3]</td>
</tr>
<tr>
<td>4</td>
<td>Complex array</td>
<td>[(1, 2), (3, 4)]</td>
</tr>
<tr>
<td>5</td>
<td>List</td>
<td>{&quot;ABC&quot;, &quot;Var&quot;}</td>
</tr>
<tr>
<td>6</td>
<td>Global name</td>
<td>X</td>
</tr>
<tr>
<td>7</td>
<td>Local name</td>
<td>y</td>
</tr>
<tr>
<td>8</td>
<td>Program</td>
<td>«A 2 +»</td>
</tr>
<tr>
<td>9</td>
<td>Algebraic</td>
<td>'X=Y^2'</td>
</tr>
<tr>
<td>10</td>
<td>Binary integer</td>
<td># 247d</td>
</tr>
<tr>
<td>11</td>
<td>Graphics object</td>
<td>Graphic 131 x 64</td>
</tr>
<tr>
<td>12</td>
<td>Tagged object</td>
<td>Dist: 84.45</td>
</tr>
<tr>
<td>13</td>
<td>Unit object</td>
<td>32 Ft/s^2</td>
</tr>
<tr>
<td>14</td>
<td>XLIB name</td>
<td>XLIB 766 1</td>
</tr>
<tr>
<td>15</td>
<td>Directory</td>
<td>DIR ... END</td>
</tr>
<tr>
<td>16</td>
<td>Library</td>
<td>Library 766: ...</td>
</tr>
<tr>
<td>17</td>
<td>Backup object</td>
<td>Backup HOMEDIR</td>
</tr>
<tr>
<td>18</td>
<td>Built-in function</td>
<td>SIN</td>
</tr>
<tr>
<td>19</td>
<td>Built-in command</td>
<td>SWAP</td>
</tr>
<tr>
<td>26</td>
<td>Library Data</td>
<td>Library Data</td>
</tr>
</tbody>
</table>
# Character Codes

<table>
<thead>
<tr>
<th>NUM</th>
<th>CHR</th>
<th>NUM</th>
<th>CHR</th>
<th>NUM</th>
<th>CHR</th>
<th>NUM</th>
<th>CHR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>■</td>
<td>32</td>
<td>_</td>
<td>64</td>
<td>@</td>
<td>96</td>
<td>'</td>
</tr>
<tr>
<td>1</td>
<td>■</td>
<td>33</td>
<td>!</td>
<td>65</td>
<td>A</td>
<td>97</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>■</td>
<td>34</td>
<td>&quot;</td>
<td>66</td>
<td>B</td>
<td>98</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>■</td>
<td>35</td>
<td>#</td>
<td>67</td>
<td>C</td>
<td>99</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>■</td>
<td>36</td>
<td>$</td>
<td>68</td>
<td>D</td>
<td>100</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>■</td>
<td>37</td>
<td>%</td>
<td>69</td>
<td>E</td>
<td>101</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>■</td>
<td>38</td>
<td>&amp;</td>
<td>70</td>
<td>F</td>
<td>102</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>■</td>
<td>39</td>
<td>'</td>
<td>71</td>
<td>G</td>
<td>103</td>
<td>g</td>
</tr>
<tr>
<td>8</td>
<td>■</td>
<td>40</td>
<td>&lt;</td>
<td>72</td>
<td>H</td>
<td>104</td>
<td>h</td>
</tr>
<tr>
<td>9</td>
<td>■</td>
<td>41</td>
<td>&gt;</td>
<td>73</td>
<td>I</td>
<td>105</td>
<td>i</td>
</tr>
<tr>
<td>10</td>
<td>■</td>
<td>42</td>
<td>*</td>
<td>74</td>
<td>J</td>
<td>106</td>
<td>j</td>
</tr>
<tr>
<td>11</td>
<td>■</td>
<td>43</td>
<td>+</td>
<td>75</td>
<td>K</td>
<td>107</td>
<td>k</td>
</tr>
<tr>
<td>12</td>
<td>■</td>
<td>44</td>
<td>,</td>
<td>76</td>
<td>L</td>
<td>108</td>
<td>l</td>
</tr>
<tr>
<td>13</td>
<td>■</td>
<td>45</td>
<td>-</td>
<td>77</td>
<td>M</td>
<td>109</td>
<td>m</td>
</tr>
<tr>
<td>14</td>
<td>■</td>
<td>46</td>
<td>.</td>
<td>78</td>
<td>N</td>
<td>110</td>
<td>n</td>
</tr>
<tr>
<td>15</td>
<td>■</td>
<td>47</td>
<td>/</td>
<td>79</td>
<td>O</td>
<td>111</td>
<td>o</td>
</tr>
<tr>
<td>16</td>
<td>■</td>
<td>48</td>
<td>0</td>
<td>80</td>
<td>P</td>
<td>112</td>
<td>p</td>
</tr>
<tr>
<td>17</td>
<td>■</td>
<td>49</td>
<td>1</td>
<td>81</td>
<td>Q</td>
<td>113</td>
<td>q</td>
</tr>
<tr>
<td>18</td>
<td>■</td>
<td>50</td>
<td>2</td>
<td>82</td>
<td>R</td>
<td>114</td>
<td>r</td>
</tr>
<tr>
<td>19</td>
<td>■</td>
<td>51</td>
<td>3</td>
<td>83</td>
<td>S</td>
<td>115</td>
<td>s</td>
</tr>
<tr>
<td>20</td>
<td>■</td>
<td>52</td>
<td>4</td>
<td>84</td>
<td>T</td>
<td>116</td>
<td>t</td>
</tr>
<tr>
<td>21</td>
<td>■</td>
<td>53</td>
<td>5</td>
<td>85</td>
<td>U</td>
<td>117</td>
<td>u</td>
</tr>
<tr>
<td>22</td>
<td>■</td>
<td>54</td>
<td>6</td>
<td>86</td>
<td>V</td>
<td>118</td>
<td>v</td>
</tr>
<tr>
<td>23</td>
<td>■</td>
<td>55</td>
<td>7</td>
<td>87</td>
<td>W</td>
<td>119</td>
<td>w</td>
</tr>
<tr>
<td>24</td>
<td>■</td>
<td>56</td>
<td>8</td>
<td>88</td>
<td>X</td>
<td>120</td>
<td>x</td>
</tr>
<tr>
<td>25</td>
<td>■</td>
<td>57</td>
<td>9</td>
<td>89</td>
<td>Y</td>
<td>121</td>
<td>y</td>
</tr>
<tr>
<td>26</td>
<td>■</td>
<td>58</td>
<td>:</td>
<td>90</td>
<td>Z</td>
<td>122</td>
<td>z</td>
</tr>
<tr>
<td>27</td>
<td>■</td>
<td>59</td>
<td>;</td>
<td>91</td>
<td>[</td>
<td>123</td>
<td>c</td>
</tr>
<tr>
<td>28</td>
<td>■</td>
<td>60</td>
<td>&lt;</td>
<td>92</td>
<td>\</td>
<td>124</td>
<td>l</td>
</tr>
<tr>
<td>29</td>
<td>■</td>
<td>61</td>
<td>=</td>
<td>93</td>
<td>]</td>
<td>125</td>
<td>)</td>
</tr>
<tr>
<td>30</td>
<td>■</td>
<td>62</td>
<td>&gt;</td>
<td>94</td>
<td>^</td>
<td>126</td>
<td>`</td>
</tr>
<tr>
<td>31</td>
<td>...</td>
<td>63</td>
<td>?</td>
<td>95</td>
<td>_</td>
<td>127</td>
<td>\</td>
</tr>
<tr>
<td>NUM</td>
<td>CHR</td>
<td>NUM</td>
<td>CHR</td>
<td>NUM</td>
<td>CHR</td>
<td>NUM</td>
<td>CHR</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>128</td>
<td>(^{\wedge})</td>
<td>160</td>
<td>(\wedge)</td>
<td>192</td>
<td>(\wedge)</td>
<td>224</td>
<td>(\wedge)</td>
</tr>
<tr>
<td>129</td>
<td>(^{\wedge}\wedge)</td>
<td>161</td>
<td>(i)</td>
<td>193</td>
<td>(i)</td>
<td>225</td>
<td>(i)</td>
</tr>
<tr>
<td>130</td>
<td>(^{\wedge}\wedge\wedge\wedge)</td>
<td>162</td>
<td>(\phi)</td>
<td>194</td>
<td>(\phi)</td>
<td>226</td>
<td>(\phi)</td>
</tr>
<tr>
<td>131</td>
<td>(^{\wedge}\wedge\wedge\wedge\wedge\wedge)</td>
<td>163</td>
<td>(\chi)</td>
<td>195</td>
<td>(\chi)</td>
<td>227</td>
<td>(\chi)</td>
</tr>
<tr>
<td>132</td>
<td>(^{\wedge}\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge)</td>
<td>164</td>
<td>(\psi)</td>
<td>196</td>
<td>(\psi)</td>
<td>228</td>
<td>(\psi)</td>
</tr>
<tr>
<td>133</td>
<td>(^{\wedge}\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wedge\wage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>NUM</th>
<th>HP 48</th>
<th>ASCII</th>
<th>NUM</th>
<th>HP 48</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>₣</td>
<td>(&lt;)</td>
<td>147</td>
<td>ε</td>
<td>(Ge)</td>
</tr>
<tr>
<td>129</td>
<td>¥</td>
<td>(x)</td>
<td>148</td>
<td>η</td>
<td>(Gn)</td>
</tr>
<tr>
<td>130</td>
<td>₣</td>
<td>.V</td>
<td>149</td>
<td>θ</td>
<td>(Gh)</td>
</tr>
<tr>
<td>131</td>
<td>$</td>
<td>(v/)</td>
<td>150</td>
<td>h</td>
<td>(Gl)</td>
</tr>
<tr>
<td>132</td>
<td>¥</td>
<td>.S</td>
<td>151</td>
<td>p</td>
<td>(Gr)</td>
</tr>
<tr>
<td>133</td>
<td>Σ</td>
<td>\GS</td>
<td>152</td>
<td>σ</td>
<td>(Gs)</td>
</tr>
<tr>
<td>134</td>
<td>▷</td>
<td>(</td>
<td>&gt;</td>
<td></td>
<td>153</td>
</tr>
<tr>
<td>135</td>
<td>π</td>
<td>\pi</td>
<td>154</td>
<td>o</td>
<td>(Gw)</td>
</tr>
<tr>
<td>136</td>
<td>ò</td>
<td>.d</td>
<td>155</td>
<td>Δ</td>
<td>(GD)</td>
</tr>
<tr>
<td>137</td>
<td>°</td>
<td>(&lt;=)</td>
<td>156</td>
<td>Π</td>
<td>(PI)</td>
</tr>
<tr>
<td>138</td>
<td>±</td>
<td>(&gt;)</td>
<td>157</td>
<td>Ω</td>
<td>(GW)</td>
</tr>
<tr>
<td>139</td>
<td>×</td>
<td>=/\</td>
<td>158</td>
<td>•</td>
<td>(]])</td>
</tr>
<tr>
<td>140</td>
<td>@</td>
<td>(Ga)</td>
<td>159</td>
<td>»</td>
<td>(oo)</td>
</tr>
<tr>
<td>141</td>
<td>-</td>
<td>(-&gt;)</td>
<td>171</td>
<td>≪</td>
<td>(&lt;&lt;)</td>
</tr>
<tr>
<td>142</td>
<td>≪</td>
<td>(&lt;)</td>
<td>176</td>
<td>≼</td>
<td>(^o)</td>
</tr>
<tr>
<td>143</td>
<td>■</td>
<td>(v)</td>
<td>181</td>
<td>μ</td>
<td>(Gm)</td>
</tr>
<tr>
<td>144</td>
<td>⤇</td>
<td>(^)</td>
<td>187</td>
<td>»</td>
<td>(&gt;&gt;)</td>
</tr>
<tr>
<td>145</td>
<td>¥</td>
<td>(Gg)</td>
<td>215</td>
<td>≏</td>
<td>(x)</td>
</tr>
<tr>
<td>146</td>
<td>ø</td>
<td>(Gd)</td>
<td>216</td>
<td>ø</td>
<td>(O/)</td>
</tr>
<tr>
<td>247</td>
<td>÷</td>
<td>(:-)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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, except for the wordsize (flags -5 → -10).

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
<th>Clear</th>
<th>Set</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Principal Solution</td>
<td>General solutions</td>
<td>Principal solutions</td>
<td>Clear</td>
</tr>
<tr>
<td>-2</td>
<td>Symbolic Constants</td>
<td>Symbolic form</td>
<td>Numeric form</td>
<td>Clear</td>
</tr>
<tr>
<td>-3</td>
<td>Numeric Results</td>
<td>Symbolic results</td>
<td>Numeric results</td>
<td>Clear</td>
</tr>
<tr>
<td>-4</td>
<td>Not used.</td>
<td></td>
<td></td>
<td>Clear</td>
</tr>
</tbody>
</table>

Symbolic Math Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
<th>Clear</th>
<th>Set</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>-5</td>
<td>Binary integer wordsize $n + 1$: $0 \leq n \leq 63$</td>
<td>Clear</td>
<td></td>
<td>64</td>
</tr>
<tr>
<td>-10</td>
<td>Flag -10 is the most significant bit</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Binary Integer Math Flags

<table>
<thead>
<tr>
<th>Binary Integer Base</th>
<th>-11</th>
<th>-12</th>
<th>DEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC</td>
<td>Clear</td>
<td>Clear</td>
<td>DEC</td>
</tr>
<tr>
<td>BIN</td>
<td>Clear</td>
<td>Set</td>
<td>DEC</td>
</tr>
<tr>
<td>OCT</td>
<td>Set</td>
<td>Clear</td>
<td>DEC</td>
</tr>
<tr>
<td>HEX</td>
<td>Set</td>
<td>Set</td>
<td>DEC</td>
</tr>
</tbody>
</table>

-13 and -14 are not used.
<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
<th>Clear</th>
<th>Set</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Coordinate System Flags</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-15</td>
<td>Rectangular</td>
<td>Clear</td>
<td>Clear</td>
<td></td>
</tr>
<tr>
<td>and</td>
<td>Cylindrical Polar</td>
<td>Clear</td>
<td>Set</td>
<td></td>
</tr>
<tr>
<td>-16</td>
<td>Spherical Polar</td>
<td>Set</td>
<td>Set</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Trigonometric Mode Flags</strong></td>
<td>-17</td>
<td>-18</td>
<td>Degrees</td>
</tr>
<tr>
<td>-17</td>
<td>Degrees</td>
<td>Clear</td>
<td>Clear</td>
<td></td>
</tr>
<tr>
<td>and</td>
<td>Radians</td>
<td>Set</td>
<td>Clear</td>
<td></td>
</tr>
<tr>
<td>-18</td>
<td>Grads</td>
<td>Clear</td>
<td>Set</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Math Exception Flags</strong></td>
<td>-19</td>
<td>-20</td>
<td></td>
</tr>
<tr>
<td>-19</td>
<td>Vector/complex</td>
<td>Vector</td>
<td>Complex</td>
<td>Vector</td>
</tr>
<tr>
<td>-20</td>
<td>Underflow Exception</td>
<td>Return 0, set -23 or -24</td>
<td>Error</td>
<td>Clear</td>
</tr>
<tr>
<td></td>
<td>Overflow Exception</td>
<td>Return ±MAXR, set -25</td>
<td>Error</td>
<td>Clear</td>
</tr>
<tr>
<td>-22</td>
<td>Infinite Result</td>
<td>Error</td>
<td>Return ±MAXR, set -26</td>
<td>Error</td>
</tr>
<tr>
<td>-23</td>
<td>Pos. Underflow Ind.</td>
<td>No Exception</td>
<td>Exception</td>
<td>Clear</td>
</tr>
<tr>
<td>-24</td>
<td>Neg. Underflow Ind.</td>
<td>No Exception</td>
<td>Exception</td>
<td>Clear</td>
</tr>
<tr>
<td>-25</td>
<td>Overflow Indicator</td>
<td>No Exception</td>
<td>Exception</td>
<td>Clear</td>
</tr>
<tr>
<td>-26</td>
<td>Infinite Result Ind.</td>
<td>No Exception</td>
<td>Exception</td>
<td>Clear</td>
</tr>
<tr>
<td></td>
<td><strong>Plotting and Graphics Flags</strong></td>
<td>-30</td>
<td>-31</td>
<td></td>
</tr>
<tr>
<td>-30</td>
<td>Function Plotting</td>
<td>f(x)</td>
<td>y and f(x)</td>
<td>f(x)</td>
</tr>
<tr>
<td>-31</td>
<td>Curve Filling</td>
<td>Filling Enabled</td>
<td>Filling Disabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>-32</td>
<td>Graphics Cursor</td>
<td>Visible Light Bkgnd</td>
<td>Visible Dark Bkgnd</td>
<td>Light</td>
</tr>
</tbody>
</table>

-27 through -29 are not used.
<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
<th>Clear</th>
<th>Set</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>-33</td>
<td>I/O Device</td>
<td>Serial</td>
<td>IR</td>
<td>Serial</td>
</tr>
<tr>
<td>-34</td>
<td>Printing Device</td>
<td>IR</td>
<td>Serial</td>
<td>IR</td>
</tr>
<tr>
<td>-35</td>
<td>I/O Data Format</td>
<td>ASCII</td>
<td>Binary</td>
<td>ASCII</td>
</tr>
<tr>
<td>-36</td>
<td>RECV Overwrite</td>
<td>New variable</td>
<td>Overwrite</td>
<td>New</td>
</tr>
<tr>
<td>-37</td>
<td>Double - Spaced Print</td>
<td>Single</td>
<td>Double</td>
<td>Single</td>
</tr>
<tr>
<td>-38</td>
<td>Linefeed</td>
<td>Inserts LF</td>
<td>Suppresses LF</td>
<td>Inserts</td>
</tr>
<tr>
<td>-39</td>
<td>Kermit Messages</td>
<td>Msg Displayed</td>
<td>Msg Suppressed</td>
<td>Displayed</td>
</tr>
</tbody>
</table>

**Time Management Flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
<th>Clear</th>
<th>Set</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>-40</td>
<td>Clock Display</td>
<td>TIME menu only</td>
<td>All times</td>
<td>TIME menu</td>
</tr>
<tr>
<td>-41</td>
<td>Clock Format</td>
<td>12 hour</td>
<td>24 hour</td>
<td>12 hour</td>
</tr>
<tr>
<td>-42</td>
<td>Date Format</td>
<td>MM/DD/YY</td>
<td>DD.MM.YY</td>
<td>MM/DD/YY</td>
</tr>
<tr>
<td>-43</td>
<td>Rpt. Alarm Reschedule</td>
<td>Rescheduled</td>
<td>Not Rescheduled</td>
<td>Rescheduled</td>
</tr>
<tr>
<td>-44</td>
<td>Acknowledged Alarms</td>
<td>Deleted</td>
<td>Saved</td>
<td>Deleted</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
<th>Clear</th>
<th>Set</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>-45</td>
<td>Set the number of digits in Fix, Scientific, and Engineering Modes</td>
<td>Clear</td>
<td>Clear</td>
<td>STD</td>
</tr>
<tr>
<td>-48</td>
<td>Engineering Modes</td>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number Display Format</th>
<th>-49</th>
<th>-50</th>
<th>STD</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>Clear</td>
<td>Clear</td>
<td>Clear</td>
</tr>
<tr>
<td>FIX</td>
<td>Clear</td>
<td>Set</td>
<td>Set</td>
</tr>
<tr>
<td>SCI</td>
<td>Set</td>
<td>Clear</td>
<td>Set</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
<th>Clear</th>
<th>Set</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>-51</td>
<td>Fraction Mark</td>
<td>Decimal</td>
<td>Comma</td>
<td>Decimal</td>
</tr>
<tr>
<td>-52</td>
<td>Single Line Display</td>
<td>Multi-line</td>
<td>Single-line</td>
<td>Multi-line</td>
</tr>
<tr>
<td>-53</td>
<td>Precedence</td>
<td>() suppressed</td>
<td>() displayed</td>
<td>Suppressed</td>
</tr>
</tbody>
</table>
### Miscellaneous Flags

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

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

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
<th>Clear</th>
<th>Set</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>Units Type</td>
<td>SI units</td>
<td>English units</td>
<td>SI units</td>
</tr>
<tr>
<td>61</td>
<td>Units Usage</td>
<td>Units used</td>
<td>Units not used</td>
<td>Units used</td>
</tr>
<tr>
<td>62</td>
<td>Payment Mode</td>
<td>End mode</td>
<td>Begin mode</td>
<td>End mode</td>
</tr>
</tbody>
</table>
## Alpha Keyboard

<table>
<thead>
<tr>
<th>a, α</th>
<th>b</th>
<th>c</th>
<th>d, Δ</th>
<th>e</th>
<th>f</th>
<th>g, γ</th>
<th>h</th>
<th>η</th>
<th>i</th>
<th>j</th>
<th>k</th>
<th>l, λ</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>n</td>
<td>μ</td>
<td>Ω</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>p</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>σ</td>
<td>t</td>
<td>u</td>
<td>%</td>
<td>v</td>
<td>~</td>
<td>w</td>
<td>w</td>
<td>x</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>@</td>
<td>y</td>
<td>±</td>
<td>z</td>
<td>Π</td>
<td>!</td>
<td>i</td>
<td>?</td>
<td>?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENTER</td>
<td>Y</td>
<td>Z</td>
<td>DEL</td>
<td>←</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Additional Symbols

<table>
<thead>
<tr>
<th>LC</th>
<th>INS</th>
<th>^</th>
<th>~</th>
<th>:</th>
<th>etc.</th>
<th>(</th>
<th>)</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>÷</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$</td>
<td>€</td>
<td>¥</td>
<td>Ξ</td>
<td>°</td>
<td>[</td>
<td>]</td>
<td>_</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>≠</td>
<td>&lt;</td>
<td>&gt;</td>
<td>≤</td>
<td>≥</td>
<td>«</td>
<td>»</td>
<td>&quot;</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONT</td>
<td>OFF</td>
<td>=</td>
<td>→</td>
<td>,</td>
<td>←</td>
<td>π</td>
<td>Δ</td>
<td>{</td>
</tr>
<tr>
<td>ON</td>
<td>0</td>
<td>•</td>
<td>SPC</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The HP 48
Programmer's ToolKit

Getting Started 1
Character Set Catalog 5
Menu Label Builder 7
Flag Catalog 10
Data Browser 13
Title Browser 22
Tool Library 26
Command Reference 40
Object Types 116
Character Codes 117
Character Translations 119
Flags 120
Alpha Keyboard 124
The HP 48 Programmer's ToolKit

Includes Disk for IBM-Compatible Computers

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.

- The Character Set Catalog is an interactive catalog that displays the HP 48 character set in all three available fonts and shows the I/O translation forms.

- The Menu Label Builder is an interactive program for building graphics objects for use in custom menus.

- The Flag Catalog is an interactive program that shows the states and descriptions of system and user flags.

- The Data Browser and Title Browser are two powerful screen-oriented user interface utilities that may be used to enhance an application:

- The Tool Library provides 74 new commands for arrays, graphics, list utilities, meta-objects, sorting, sets, stack utilities, string utilities, and more. These commands are fast and powerful. For instance, the command MREVERSE will reverse the order of 1000 objects on the stack in less than a second, and ROTATE will rotate character strings, lists, or graphics objects.

The 124 page manual provides complete descriptions of all the new catalogs, programs, and commands with examples. Additional chapters provide reference tables for object types, the character set, and flags.

Armstrong Publishing Company
3135 N.W. Ashwood Drive Corvallis, OR 97330 USA