HP48 SX SmartROM ™

User's Guide

This manual was produced with Ventura Publisher© and a CANON BJ10e® Bubble Jet Printer.

Printed in Italy

Foreword

Thank you for the purchase of the SmartROM. You bought a quality software that will greatily help in your work, hobby or study!

If you are new to the 48's world, we suggest you read carefully the User's manual before diving into deep waters of the SmartROM.

The well-tempered 48

The SmartROM plays the 48 like no other product did before. Of course it is not perfect, but as you will experiment on your own, it will become difficult to do something without it. Its huge set of housekeeping hidden commands lets you solve problems that you hardly could imagine.

The SmartROM is designed to maximize programmer productivity by concentrating a lot of powerful commands and utilities in one package and setting a standard in some areas like symbolic matrix handling and meta objects manipulation. If you are an RPL enthusiast, you will appreciate SmartROM at most, reminiscing its venerable predecessors like the ZenROM and the PPCRom for the HP-41 and the JPCRom for the HP-71. For the joy of hackers we left a lot of commands in the dark. Sometimes we turn on the light at the right spot. Hidden commands have been created for future applications and represent a 40% of the whole ROM. We expect you dig deeply in the ROM to find them. If you get tired digging in the ROM, you may refer to the *Hidden Commands Reference.* As you will see later, some utilities on the disk use intensively SmartROM's hidden functions. This manual refers to version 1:B of the SmartROM wich fixes known bugs in the previous version, adds many new hidden functions and extends some commands as well. We hope also you will appreciate our efforts of keeping things as simple as possible, sacrificing sometimes speed to flexibility and expandibility. The concept of expandibility is a cornerstone of the SmartROM and represents, in our opinion, its first strong point. Second strong point is the Symbolic Matrix Writer which is the most complex application available for the SmartROM and one of the most useful tools available for the calculator.

For those who need an in-depth knowledge of the SmartROM, a Hidden Commands Reference Manual (in English) is available.

All the information contained herein, when a different source is not specified, come from our own experiments on the calculator and are proprietary. Appendix C, containing Saturn Assembly Language description, complies with the description given in the HP-71 IDS Vol. I-II copyrighted by Hewlett-Packard.

Manual's Contents

Chapter 1: SmartROM Commands Reference

This is the reference section of the manual covering the entire set of commands listed in alphabetical order. For each command a stack diagram is given and remarkable information as well. Examples are collected at the end of each paragraph.

Appendix A: Warranty, Service and Support

Refer to this section when you encounter problems using the SmartROM.

Appendix B: Objects

HP-48 objects structure is explained in detail.

Appendix C: The Saturn Microprocessor

A description of the CPU and its machine language instruction set is given.

Appendix D: System calls

Most used system entry points are listed in address order.

Appendix E: Error Messages

Contains the complete list of the errors generated by the SmartROM along with possible causes.

Appendix H: Hidden Commands

Contains the exhaustive list of SmartROM hidden commands version 1:B, subdivided in logic categories.

List of SmartROM RPL commands.

This is the complete list of SmartROM RPL commands subdivided in logic categories. Some commands appear in several categories being polymorphic functions.

	Name
Stack Manipulation	AAB
	BAA
	BAB
	BBA
	BCAC
	BCDA
	C2M
	CAB
	CBA
	MARK
	NIP
	RDOWN
	RDROP
	RDUP
	RUP
	SHIFT
	XLVLS
Argument Checking	CHL?
	CHSET?
	CHST?
List Manipulation	FIND
	L2M
	LOP1
	LOPN
	LVOP
	NULL
	REPLACE
	REV
	RPT
	SPLIT
	SRDIFF
	SRGE
	SRGT
	SRLE
	SRLT
String Manipulation	→Char
	FIND
	LINES→
	→LINES
	LOC
	MEMBER
	NULL
	PARSE
	REPLACE
	ROWCOL

	REV
	RPT
	SPAN
	SPLIT
Meta Object Manipulation	COPY
	DELETE
	LINES→
	→LINES
	META
	METOP
	MOVE
	MREV
	NDUPN
	PKMETA
	PRG→
	→PRG
	SRT
	SRTD
Type Management	→B
	→Char
	EXT→
	→EXT
	FALSE
	MSBIT
	PRG→
	→PRG
	→R
	→SYS
	→TorF
	→Xlib
	TRUE
Symbolic Matrix Handling	ADD
	ADDCON
	CMPL
	CONFORM?
	CONST
	DELCOL
	DELROW
	DETERM
	DIMS
	EQUAL?
	FACTOR
	IDNT
	MATWRT
	MGET
	MPUT
	MSYMB?
	MULT
	SQUARE?
	SUBT
	SYMBMAT→
	→SYMBMAT
	TRNSP

Program Editing	FIND PRG→ →PRG REPLACE
O	KEIWAIT
Utility	CSTMENU RPT
ROM Revision	ROMV VER$

List of applications

This is the list of available applications stored on the diskette. INSTALL helps you installing the SmartROM by copying some auxiliary programs for the command PKMETA in the HOME directory and redefining the variable CST for a quick access to commands. Original value of CST is pushed on the stack. For this reason INSTALLshould be used only at installation time. Applications listed below are stored on the diskette along with their documentation.

Name

CALENDAR
FFT
LWC
UPC
MATMENU
PRSYMB
PROBJ
PRTHREAD
L→TH
TH→L
SHRINK
UPTRIM
PIE
BARPLOT2
INSTALL
INVRT
→FONT
*alfa*ORDER
POPDIR
CGINDIR
REPLP
CGXLIB
XREF

Typefaces conventions

DISPLAY	Used to represent text as it appear on the display or anything you must type.
Italics	Italics are used to introduce a new term or to emphasize words or sentences.
[KEY]	Keys are surrounded by square brackets.
LABEL	This typeface represents a menu label.

Typographic conventions

The special set of characters implemented in the 48 requires a special treatment in order to avoid misunderstanding. Greek characters are represented as their name in superscript mode. Below there is a table summarizing the conventions used throughout the manual.

#b	Binary integer #12345h, #01101b, etc.
External	System Address.
<h>	System Binary.
n, d, i, j, k, l,	Real numbers.
TRUE	System boolean (External).
FALSE	System boolean (External).
Complex	Complex number .
'NAME'	A global name.
'name'	A local name.
obj	Any object.
Symb	Symbolic Expressions, real and complex numbers, units, gloabal or local names.
prg	RPL program
«...»	User RPL Program.
str	String of characters.
Char	Character object
[...]	Array (Vector).
[[...]...[...]]	Array (Matrix).
{...} or List	List.
{{$Symb_{11}$ ... $Symb_{1n}$} {$Symb_{m1}$ ... $Symb_{mn}$}}	Symbolic Matrix.
obj_1...obj_n n	Any Meta-object.
str_1...str_n n	String Meta-object.
XLIB LID Num	External Library name.

Automatic Installation

The installation of the library is, under normal circumstances, completely transparent to the user, who must only plug the card in the slot. HP-48 user's manual describes well this procedure in chapter 34, Volume II.

Note that the configuration program of the library always attaches the libraries in the HOME directory. If you want to attach manually the library in a specified subdirectory, you must follow the procedure described in Chapter 34 of the HP-48 User's Manual. However if a warm start happens, the library will be reattached to the HOME directory.

After this stage, you can verify the installation of the SmartROM by following the procedure described below:

1) Type HOME **[ENTER]**, then **[Gold] LIBRARY**: now, if you installed the card in Port 1 two new labels called **MATW** and **SMART** should appear in the menu. May be you need to press **[NXT]** to advance menu pages. The order in which labels appear depends on the number of libraries residing in memory. If you see these labels, go on to stage 5.

2) If you are not able to see any new label, may be you have not correctly plugged in the card. Repeat stage 1.

3) If the label does not appear, try to change port and start over again.

4) If labels do not appear yet, your card requires service. See in the Appendix A the details about Service and Warranty.

5) Type in the command line: &:INSTALL **[EVAL]**

   After some seconds, a copy of your CST variable should appear on the stack along with a custom menu.

6) The library is ready for use.

Manual Installation

Type HOME **ENTER**, :&:INSTALL **EVAL** . If the calculator does not report errors and the value of CST appear on the stack, the library has been successfully installed, otherwise check the list below:

ERROR	CAUSE & REMEDY
Port Not Available	The card is not properly installed or is bad. Try to reinstall the card or change slot.
none, but CST was not pushed on stack.	May be you have an INSTALL program overriding SmartROM's one. Cancel it.
Directory not allowed	You have a directory whose name is conflicting with the subprogram being copied in the HOME directory. Remove or rename the directory.

SmartROM Commands Reference

This section explains in detail each command of the ROM, giving its stack representation, remarkable information, examples and names of applications containing the command. We strongly suggest you edit our applications and improve them with your own customization. If you have any question about SmartROM commands, contact us at the address given in Appendix A.

All SmartROM commands obey to the rules of HP-48 system commands, preserving stack contents. Symbolic matrix commands, do not preserve correctly LASTARG parameters because we used standard math operators during the calculations. When a calculation goes on, each operator saves its arguments for a subsequent LASTARG, effectively overriding previously saved parameters. Thus, if you try a LASTARG after a symbolic matrix command, resulting parameters reflect the stack arguments at the time of the last arithmetic operation. Nevertheless, after the execution from the keyboard of such a command the LAST STACK toggle (if enabled) works correctly and the original stack may be recovered.

All the commands are accessible via SmartROM's Custom menu once you invoke the command CSTMENU or you execute the Backup program INSTALL, as described in the previous chapter.

All the commands contained in this section belong to Library 821 (Smart) except MATWRT which belongs to Library 822 (MATLIB).

Library 821, 822 and 823 have been allocated officially by Hewlett-Packard for Smart Technology's Products.

They cannot be used by other commercial software developers.

If you remove the card from the HP-48SX, SmartROM commands will appear as XLIB 821 nnn, where nnn is a number between 0 and 251. Visible commands occupy the range 0-95. All other commands are hidden and have no correspondent text. If you want to know what hidden commands do, download the Hidden Commands Reference from the BBS and print it by yourself. Otherwise ask your dealer for a copy.

Hidden Commands contained in libraries other than 821 are not supported and shall not be used. Any future extension to SmartROM will use Library 822.

AAB

Category	Stack Manipulation
Affected by flag	none
Input	2: Obj_A 1: Obj_B
Output	3: Obj_A 2: Obj_A 1: Obj_B
Function	Duplicates the object on the second level.
See Also	BAA, BAB, BBA, BCAC, BCDA, CAB, CBA, NIP

ADD

Category Symbolic Matrix Manipulation

Affected by flag -3 (Symbolic result)

Input

2: {{SymbA$_{1,1}$ SymbA$_{1,2}$ ... SymbA$_{1,n}$}
...
{SymbA$_{m,1}$ SymbA$_{m,2}$ ... SymbA$_{m,n}$}}
1: {{SymbB$_{1,1}$ SymbB$_{1,2}$ ... SymbB$_{1,n}$}
...
{SymbB$_{m,1}$ SymbB$_{m,2}$ ... SymbB$_{m,n}$}}

Output

1: {{SymbA$_{1,1}$+SymbB$_{1,1}$ SymbA$_{1,2}$+SymbB$_{1,2}$ ... SymbA$_{1,n}$+SymbB$_{1,n}$}
...
{SymbA$_{m,1}$+SymbB$_{m,1}$ SymbA$_{m,2}$+SymbB$_{m,2}$ ... SymbA$_{m,n}$+SymbB$_{m,n}$}}

Function Performs the addition of two symbolic arrays.

Notes A one-dimensional array must be written as one-row or one-column two-dimensional array. To avoid run-time errors the flag -3 must be clear, or, alternatively, each symbolic expression must evaluate to a numeric value.

The routine does not check for symbolic values. The reason is explained below:

the routine makes use of a hidden command which allows to pass the operator unevaluated. Moreover the operator is not restricted only to symbolic objects but can work on all pairs of objects for which the operation is meaningful. You could even make a list-array of real arrays and perform the addition cell by cell between each pair of arrays. See the example below.

See Also ADDCON, DIMS, EQUAL?, MATWRT, MULT, MSYMB?, SUBT

Applications MATMENU, PRSYMB

Examples

```
2:  {{ 1 2  3 } {  'X'   'X+1' 'X-3' }}
1:  {{-1 2 -3 } {'-X-1' 'X+1' '3-X' }}
```

ADD **[ENTER]**

```
1:  {{ 0 4 0 } {-1 '2+2*X' 0 }}
```

```
2:  {{ [ 1  2  3] [ 4 5 6 ] }}
1:  {{ [-1 -2 -3 ] [ 1 2 3 ] }}
```

ADD **[ENTER]**

```
1:  {{ [ 0 0 0 ] [ 5 7 9 ] }}
```

ADDCON

Category Symbolic Matrix Manipulation

Affected by flag -3 (Symbolic Result)

Input

2: $\{\{ Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,n} \}$
$\ldots$
$\{ Symb_{m,1}\ Symb_{m,2} \ldots Symb_{m,n}\}\}$

1: Symb

Output

2: $\{\{Symb+Symb_{1,1}\ Symb+Symb_{1,2} \ldots Symb+Symb_{1,n} \}$
$\ldots$
$\{Symb+Symb_{m,1}\ Symb+Symb_{m,2} \ldots Symb+Symb_{m,n} \}\}$

Function Adds a constant value to all the elements of a list-matrix.

Notes See notes about flag under ADD.

See Also ADD, FACTOR, MATWRT, MSYMB?

Applications MATMENU, PRSYMB

Examples

```
2:          {{ 1     2      3}
              {'X' 'X+1' 'X-3'}}
1:          '-X-1'
```

ADDCON **[ENTER]**

```
1:          {{'-X' '1-X' '2-X'}
              { -1   0    -4  }}
```

→B

Category Type Conversion

Affected by flag -5 to -10 (Binary Wordsize) and -11 to -12 (Binary Integer Base)

Input

1: h 1: Char 1: <h>

Output

1: #h

Function Converts a numeric value into a binary integer. Extended precision numbers are not allowed. Pay attention to the current wordsize that may truncate the result.

Notes →B allows binary integers as input. This technique is useful to set the user free of inputting data in the preferred form.

See Also EXT→, →EXT, →R, →SYS

Example

```
5:  30
4:  #20h
3:  <13h>
2:  1.5
1:  4
```

{ →B } METOP **[ENTER]**

```
5:  #1Eh
4:  #20h
3:  #13h
2:  #2h
1:  4
```

BAA

Category	Stack Manipulation
Affected by flag	none
Input	2: Obj_A 1: Obj_B
Output	3: Obj_B 2: Obj_A 1: Obj_A
Function	Swaps the objects and duplicates the first level.
See Also	AAB, BAB, BBA, BCAC, BCDA, CAB, CBA, NIP

BAB

Category	Stack Manipulation
Affected by flag	none
Input	2: Obj_A 1: Obj_B
Output	3: Obj_B 2: Obj_A 1: Obj_B
Function	Duplicates the object on the first level and inserts it above the second level.
See Also	AAB, BAA, BBA, BCAC, BCDA, CAB, CBA, NIP

BBA

Category	Stack Manipulation
Affected by flag	none
Input	2: Obj_A 1: Obj_B
Output	3: Obj_B 2: Obj_B 1: Obj_A
Function	Duplicates the object on the first level and rotates the first three levels.
See Also	AAB, BAA, BAB, BCAC, BCDA, CBA, NIP

BCAC

Category	Stack Manipulation
Affected by flag	none
Input	3: Obj_A 2: Obj_B 1: Obj_C
Output	4: Obj_B 3: Obj_C 2: Obj_A 1: Obj_C
Function	Rotates the first three levels and duplicates the second level.
See Also	AAB, BAA, BAB, BBA, BCDA, CAB, CBA, NIP

BCDA

Category	Stack Manipulation
Affected by flag	none
Input	4: Obj_A 3: Obj_B 2: Obj_C 1: Obj_D
Output	4: Obj_B 3: Obj_C 2: Obj_D 1: Obj_A
Function	Rotates first four levels.
See Also	AAB, BAA, BAB, BBA, BCAC, CAB, CBA, NIP

C2M

Category Stack Manipulation and Meta-object Manipulation

Affected by flag none

Input

N+1	"MARK'
N:	Obj_1
....:	...
2:	Obj_{n-1}
1:	Obj_n

Output

N+2:	"MARK'
N+1:	Obj_1
....:	...
2:	Obj_n
1:	n

Function Counts the elements between the mark and the top of the stack. C2M allows you to convert an unbound meta-object to a fixed size meta-object. The mark is not removed from the stack after the count. Unbound meta-objects are very useful when you don't know in advance the number of objects you shall play with.

Notes When the mark is missing, C2M is the same as DEPTH.

'MARK is a private symbol. Hidden functions make use of an alternate mark to avoid collision with user defined programs.

See Also L2M, MARK, META

Examples

3:	"MARK'
2:	10
1:	20

C2M **[ENTER]**

4:	"MARK'
3:	10
2:	20
1:	2

CAB

Category	Stack Manipulation
Affected by flag	none
Input	3: Obj_A 2: Obj_B 1: Obj_C
Output	3: Obj_C 2: Obj_A 1: Obj_B
Function	Rotates backwards the first three levels.
See Also	AAB, BAA, BAB, BBA, BCAC, BCDA, CBA, NIP

CBA

Category	Stack Manipulation
Affected by flag	none
Input	3: Obj_A 2: Obj_B 1: Obj_C
Output	3: Obj_C 2: Obj_B 1: Obj_A
Function	Reverses the order of the first three levels.
See Also	AAB, BAA, BAB, BBA, BCAC, BCDA, CAB, NIP

CHL?

Category Argument Checking

Affected by flag none

Input

2: { obj_1 obj_2 ... obj_p }
1: { t_1 t_2 ... t_n }

Output

3: { obj_1 obj_2 ... obj_p }
2: { err_1 err_2 ... err_k }
1: 1

2: { obj_1 obj_2 ... obj_p }
1: 0

Function

Checks if the elements of a list comply with the types specified. Each element of the list on the first level can be a real value (rounded to an integer) or a list of reals. Argument checking is strictly positionalin CHL?. If you need non-positional argument checking, use CHSET?.

Type Numbers follow the classification given by the command TYPE.

When a sublist of reals is specified, a multiple choice is allowed on that position of the input list.

The command return a boolean flag indicating success when the flag is 0 and a fault when the flag is 1 plus a list of mismatching positions.

The underlying meta-object structure of the result unifies the boolean convention with the meta-object convention. The output is especially suitable for IFT or IFTE input.

See Also CHSET?, CHST?

Applications MATMENU

Examples

2: { 1 2 "" }
1: { 0 0 2 }

CHL? **[ENTER]**

2: { 1 2 "" }
1: 0

2: { 1 2 "" }
1: { 0 0 { 0 1 } }

CHL? **[ENTER]**

3: { 1 2 "" }
2: { 3 }
1: 1

CHSET?

Category Argument Checking

Affected by flag none

Input

2: { obj_1 obj_2 ... obj_p }
1: { t_1 t_2 ... t_n }

Output

3:	{ obj_1 obj_2 ... obj_p }		
2:	{ obj_1 obj_2 ... obj_p }	2:	{ err_1 err_2 ... err_k }
1:	0	1:	1

Function

Checks is objects contained in a list are of the type specified, independently from the position. Each element of the list of types must be a real number (rounded to an integer). Type Numbers follow the classification given by the command TYPE. The command return a boolean flag indicating success when the flag is 0 and a fault when the flag is 1 plus a list of mismatching positions.

The underlying meta-object structure of the result unifies the boolean convention with the meta-object convention. The output is especially suitable for IFT or IFTE input.

CHSET? is useful to check if a list contains homogeneus data with minimal memory requirements.

See Also CHL?, CHST?

Examples

2: { 1 2 "" }
1: { 0 2 }

CHSET? **[ENTER]**

2: { 1 2 "" }
1: 0

2: { 1 2 "" }
1: { 0 }

CHSET? **[ENTER]**

3: { 1 2 "" }
2: { 3 }
1: 1

CHST?

Category Argument Checking

Affected by flag none

Input

N+1:	obj_1
...:	...
2:	obj_n
1:	$\{ t_1\ t_2 \ldots t_n \}$

Output

N+2:	obj_1		
...:	...	N+1:	obj_1
K :	obj_k	...:	...
...:	...	K :	obj_k
3:	obj_n	...:	...
2:	{ ... errk ...}	2:	obj_n
1:	1	1:	0

Function Checks if the elements of the stack comply with the types specified in a list. Each element of the control list on the first level can be a real value (rounded to an integer) or a list of reals. Argument checking is strictly positional in CHST?. The first element of the control list specifies the type or the set of types allowed for the object lying on the correspondent level of the stack. In effect objects on the stack are mapped as if they were collected in a list, with the element on the top that comes last.

Type Numbers follow the classification given by the command TYPE.

When a sublist of reals is specified, a multiple choice is allowed on the corresponding level of the stack. The command return a boolean flag indicating success when the flag is 0 and a fault when the flag is 1 plus a list of mismatching positions.

The underlying meta-object structure of the result unifies the boolean convention with the meta-object convention. The output is especially suitable for IFT or IFTE input.

See Also CHL?, CHSET?

Applications MATMENU

Examples

```
4:  1
3:  2
2:  ""
1:  { 0 0 2 }
```

CHST? **[ENTER]**

```
4:      1
3:      2
2:      ""
1:      0

4:      1
3:      2
2:      ""
1:      { 0 0 { 0 1 } }
```

CHST? **[ENTER]**

```
5:      1
4:      2
3:      ""
2:      { 3 }
1:      1
```

→Char

Category	Type Conversion
Affected by flag	none
Input	1: n 1: <n> 1: #n or 1: Str
Output	1: Character
Function	Converts the input object into a character object.
Notes	Character objects are used internally by the system to minimize memory storage respect to one-character strings. Often, routines taking strings as parameters allow also single characters. →CHAR accepts a character as input. This technique is useful to set the user free of inputting data in the preferred form.
See Also	→B, EXT→, →EXT, →R, →SYS

CMPL

Category Symbolic Matrix Manipulation

Affected by flag none

Input

2: {{$Symb_{1,1}$ $Symb_{1,2}$ ... $Symb_{1,n}$ }
...
{$Symb_{n,1}$ $Symb_{n,2}$ ... $Symb_{n,n}$ }}

1: { i j }

Output

1: {{$Symb_{1,1}$ ... $Symb_{1,i-1}$ $Symb_{1,i+1}$ ... $Symb_{1,n}$ }
...
{$Symb_{j-1,1}$ ... $Symb_{j-1,i-1}$ $Symb_{j-1,i+1}$ ... $Symb_{n,n}$ }
{$Symb_{j+1,1}$ ... $Symb_{j+1,i-1}$ $Symb_{j+1,i+1}$ ... $Symb_{n,n}$ }
...
{$Symb_{n,1}$ ... $Symb_{n,i-1}$ $Symb_{n,i+1}$ ... $Symb_{n,n}$ }}

Function Returns the complement of a square list-matrix, given element's subscripts. Resulting array will be of n-1 order. The array cannot be of order less than 2.

Notes In order to compute the algebraic complement of an element, the following procedure could be used.

-3 CF	Symbolic result enabled.
BAB	saves element subscripts
CMPL DETERM	computes the determinant of the minor.
SWAP OBJ→ DROP	Recalls the subscripts
+ -1 SWAP ^ *	Adjusts the sign

See Also ADD, ADDCON, CONSTMAT, DETERM, FACTOR, MATWRT, MULT, SQUARE?

Applications MATMENU

Examples

2: {{1 0 0 } {'-X' '1-X' '2-X'} {-1 0 -4}}
1: { 1 1 }

CMPL **[ENTER]**

1: {{ '1-X' '2-X'} { 0 -4}}

CONFORM?

Category Symbolic Matrix Manipulation

Affected by flag none

Input

2: $\{\{SymbA_{1,1} \ldots SymbA_{1,n}\}$ … $\{SymbA_{m,1} \ldots SymbA_{m,n}\}\}$
1: $\{\{SymbB_{1,1} \ldots SymbB_{1,p}\}$ … $\{SymbB_{n,1} \ldots SymbB_{n,p}\}\}$

2: $\{\{SymbA_{1,1} \ldots SymbA_{1,n}\}$ … $\{SymbA_{m,1} \ldots SymbA_{m,n}\}\}$
$\{\{SymbB_{1,1} \ldots SymbB_{1,q}\}$ … $\{SymbB_{p,1} \ldots SymbB_{p,q}\}\}$

Output

4: $\{\{SymbA_{1,1} \ldots SymbA_{1,n}\}$ … $\{SymbA_{m,1} \ldots SymbA_{m,n}\}\}$
3: $\{\{SymbB_{1,1} \ldots SymbB_{1,p}\}$ … $\{SymbB_{n,1} \ldots SymbB_{n,p}\}\}$
2: { m n n p }
1: 1

3: $\{\{SymbA_{1,1} \ldots SymbA_{1,n}\}$ … $\{SymbA_{m,1} \ldots SymbA_{m,n}\}\}$
2: $\{\{SymbB_{1,1} \ldots SymbB_{1,q}\}$ … $\{SymbB_{p,1} \ldots SymbB_{p,q}\}\}$
1: 0

Function Checks if the dimensions of two input matrices are suitable for row-by-column multiplication. If not a 0 is returned along with the input matrices.

Notes We call this condition conformability. When you try a row-by-column multiplication between incompatible arrays a special error is issued to inform you of this particular condition.

No check is made on the input lists with regard to their contents. If you want to check in advance list contents, use MSYMB?.

See Also DIMS, MATWRT, MSYMB?, SQUARE?

Applications MATMENU, PRSYMB

Examples

```
2:  { { 1 2 } { 'X' 'Y-2' } }
1:  { { 0 1 '-Z'} { -2 'Y' 3 } }
```

MULT **[ENTER]**

```
1:  { {     -4      '1+2*Y'        '-Z+6' }
      { '-2*(Y-2)' 'X+(Y-2)*Y' 'X*-Z+(Y-2)*3' } }
```

CONSTMAT

Category Symbolic Matrix Manipulation

Affected by flag none

Input

2: Symb
1: { m n }

Output

1: {{$Symb_{1,1}$ $Symb_{1,2}$ ... $Symb_{1,n}$}
...
{$Symb_{m,1}$ $Symb_{m,2}$ ... $Symb_{m,n}$}}

Function Returns a constant symbolic array according to the dimensions specified in the list.

See Also DIMS, MATWRT, MSYMB?

Applications MATMENU, PRSYMB

Examples

2: 'X-1'
1: { 2 3 }

CONSTMAT **[ENTER]**

1: {{ 'X-1' 'X-1' 'X-1' }
{ 'X-1' 'X-1' 'X-1' }}

COPY

Category Meta-object Manipulation

Affected by flag none

Input

....:	obj_1
....:	...
....:	obj_{from}
....:	...
....:	obj_{to}
....:	...
....:	obj_{above}
....:	...
5:	obj_n
4:	n
3:	from
2:	to
1:	above

Output

....:	obj_1
....:	...
....:	obj_{from}
....:	...
....:	obj_{to}
....:	...
....:	$obj_{above-1}$
....:	obj_{from}
....:	...
....:	obj_{to}
....:	obj_{above}
....:	...
....:	obj_n
1:	n+abs(f-t+1)

Function Copies a section of the meta-object delimited by from and to above the level above, expanding the meta-object and updating the counter.

Notes Top of stack contains the counter while level 2 contains the last object of the meta-object (as if it were the last element of a list). The fist element of the meta-object lies on level n+1.

Input parameters (from) and (to) can be given in any order. If you specify for (above) a value greater than n, the section will be appended to the tail of the meta-object.

Note that you can copy a block within itself.

A bit of theory:

Meta-objects are a metaphysical entity invented by HP people to identify a set of objects getting handled as a whole thing. They are not worth to be considered a true object unless You have the SmartROM. As you will see later, there can be string meta-objects, real meta-objects and program meta-objects as well. The most esoteric form of a meta-object is the unbound meta-object wich has no counter at all, but only a marker above its head. Internal routines of SmartROM make heavily use of meta-object utilities, because of their intrinsic compactness.

See Also DELETE, MOVE

Examples

```
6:      "TOP
5:      "these lines"
4:      "get copied"
3:      "after the bottom"
2:      "BOTTOM"
1:      5
```

2 4 6 COPY **[ENTER]**

```
9:      "TOP
8:      "these lines"
7:      "get copied"
6:      "after the bottom"
5:      "BOTTOM"
4:      "these lines"
3:      "get copied"
2:      "after the bottom"
1:      8
```

```
5:      "HEAD"
4:      "SUBHEAD"
3:      "GO OVER THE TOP"
2:      "ME TOO"
1:      4
```

3 5 1 COPY **[ENTER]**

```
7:      "GO OVER THE TOP"
6:      "ME TOO"
5:      "HEAD"
4:      "SUBHEAD"
3:      "GO OVER THE TOP"
2:      "ME TOO"
1:      6
```

```
5:      "HEAD"
4:      "EXPAND MYSELF #1"
3:      "EXPAND MYSELF #2"
2:      "BOTTOM"
1:      4
```

2 3 3 COPY **[ENTER]**

```
7:      "HEAD"
6:      "EXPAND MYSELF #1"
5:      "EXPAND MYSELF #1"
4:      "EXPAND MYSELF #2"
3:      "EXPAND MYSELF #2"
2:      "BOTTOM"
1:      6
```

CSTMENU

Category Utility

Affected by flag none

Input

Output

Function Redefines CST contents with the SmartROM custom menu (two pages) wich subdivides commands in six main areas called **STACK**, **$&L**, **META**, **SYMB**, **TYPES**, **MISC** for easier referencing and adds the following three new features:

#CHR A keytrap for entering a character given its ASCII code. The trap waits indefinitely key presses and does not allow meaningless keystrokes. Character 0 cannot be entered nor characters above 255. If you disabled the beeper, no sound comes at key press nor the jingle that welcomes you to the keytrap. The message "Enter three digits" is displayed in the status area until you press the third key. The **[ON]** key is disabled. When you enter **[2] [5]**, the keyboard is mapped to respond only to keys **[0]** - **[5]**.

VISIT Enhanced Visit Function:

Input

3: Absolute position	3: { row col }
2: 0 (replace) or 1 (insert)	2: 0 (replace) or 1 (insert)
1: 'name'	1: 'name'

3: "string"
2: occurrence
1: 'name'

This new Visit feature lets you place the cursor at a specified place in the editor, by specifying its position in three different ways. When you specify a search string and supply a 0 as occurrence, the cursor will be placed on the last occurrence of the string. If the string does not appear in the stream, the cursor will be placed at the end of the text.

SYSEVAL Syseval typing aid.

blank Intentionally left blank.Reserved for future use.

blank Intentionally left blank.Reserved for future use.

Icon The spreadsheet icon labels the Symbolic MatrixWriter trap. By pressing the rightmost key, you enter an empty MatrixWriter.

[Gold] ICON When you have a list-matrix on the first level of the stack you can edit it by pressing this keystroke, like you do with variables with the implicit STO. Once you enter in the Symbolic MatrixWriter environment, you will see old labels and new ones as well. Known labels like **→GO**, **→STK** and **WID→** still act in the usual way. **EQ W** is a new feature that allows you to pass a cell to the EquationWriter for editing. The cell cannot be empty. If you press **[ON]** while you are in the EquationWriter, the old value is restored in the cell and you will be returned to the MatrixWriterenvironment. By pressing **[ENTER]**, you validate any modification. **+ROW**, **-ROW**, **+COL**, **-COL** act in the usual way. **→ROW**, **←ROW**, **→ROW**, **↓ROW**, **→COL**, **←COL**, **↑COL**, **↓COL** rotate blocks of rows or columns. If you change menu while in the MatrixWriter, you can restore the main menu by pressing **[Blue] [ENTER]**. Within this environment you cannot enter values other than Symbolics, that is Reals, Complex, Units, Algebraics, Global names. If you try to enter objects other than these, an error will be issued.

DELCOL

Category Symbolic Matrix Manipulation

Affected by flag none

Input

2: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,n}\}$
$\vdots$
$\{Symb_{m,1}\ Symb_{m,2} \ldots Symb_{m,n}\}\}$

1: j

Output

1: $\{\{Symb_{1,1} \ldots Symb_{1,j-1}\ Symb_{1,j+1} \ldots Symb_{1,n}\}$
$\vdots$
$\{Symb_{m,1} \ldots Symb_{m,j-1}\ Symb_{m,j+1} \ldots Symb_{m,n}\}\}$

Function Deletes the specified column from the list.

Notes No restrictions on the shape of the list. If an invalid subscript is specified, an error is issued. The list-matrix can contain any object type. If you need to delete at once column and row, use CMPL.

See Also CMPL, CONSTMAT, DELROW

Examples

```
2:  {{1  0 0 } {'-X'  '1-X' '2-X'} {-1  0 -4}}
1:  1
```

DELCOL **[ENTER]**

```
1:  {{  0     0 }
     { '1-X' '2-X'}
     {  0     -4}}
```

DELETE

Category Meta-object Manipulation

Affected by flag none

Input

....:	obj_1
....:	...
....:	obj_{from}
....:	...
....:	obj_{to}
....:	...
4:	obj_n
3:	n
2:	from
1:	to

Output

....:	obj_1
....:	...
....:	obj_{from-1}
....:	obj_{to+1}
....:	...
2:	obj_n
1:	n-ABS(from-to+1)

Function Deletes a block of the meta-object, updating the counter.

Notes (from) and (to) can be given in any order. Values out of range will generate an error message. When you delete all the objects, on the stack remains only a 0 (the counter).

See Also DELETE, MOVE

Examples

```
6:  "delete after this line"
5:  "garbage"
4:  "garbage"
3:  "garbage"
2:  "last"
1:  5
```

2 4 DELETE **[ENTER]**

```
3:  "delete after this line"
2:  "last"
1:  2
```

DELROW

Category Symbolic Matrix Manipulation

Affected by flag none

Input

2: {{$Symb_{1,1}$ $Symb_{1,2}$ ... $Symb_{1,n}$ }
...
{$Symb_{m,1}$ $Symb_{m,2}$ ... $Symb_{m,n}$ }}
1: i

Output

1: {{$Symb_{1,1}$ $Symb_{1,2}$ ... $Symb_{1,n}$ }
...
{$Symb_{i-1,1}$ $Symb_{i-1,2}$ ... $Symb_{i-1,n}$}
{$Symb_{i+1,1}$ $Symb_{i+1,2}$ ... $Symb_{i+1,n}$}
...
{$Symb_{m,1}$ $Symb_{m,2}$ ... $Symb_{m,n}$ }}

Function Deletes the specified row from the list.

Notes No restrictions on the shape of the list. If an invalid subscript is specified, an error is issued. The list-matrix can contain any object type. If you need to delete at the same time a pair of crossing column and row, use CMPL.

See Also CMPL, CONSTMAT, DELCOL

Examples

2: {{1 0 0 } { '-X' '1-X' '2-X'} { -1 0 -4}}
1: 1

DELROW **[ENTER]**

1: {{ '-X' '1-X' '2-X'}
{ -1 0 -4}}

DETERM

Category Symbolic Matrix Manipulation

Affected by flag -3 (Symbolic Result)

Input

1: {{$Symb_{1,1}$ $Symb_{1,2}$ ... $Symb_{1,n}$ }
...
{$Symb_{n,1}$ $Symb_{n,2}$ ... $Symb_{n,n}$ }}

Output

1: Symbolic

Function Calculates the determinant of a square symbolic matrix.

Notes The result has not been simplified yet. You can use EXPAND and COLCT or the sample program EXCO described in Chapter 31 of the HP-48 User's Manual. The algorithm being used is described in the Hidden Commands Reference.

The algorithm is optimized and makes large use of pivoting. The precision of a numeric result varies from case to case. Sometimes is more precise than DET, while in other case is worse.

See Also SQUARE?

Applications INVRT, MATMENU, PRSYMB

Examples

1:
```
{{ 'X'   4   -1  '(X-2)/Y'}
 { 0     1   '-X'   '2*Y' }
 { 'Y/X' '1/X'  -3   'X-Y' }
 { 0     1    Y   'Y-2*X'}}
```

DETERM **[ENTER]** EXCO **[ENTER]**

```
'-2-1/X*Y^2-8/X*Y^3-2/X*Y+X*Y^2+4*X^2-X^3+12*X*Y
-2*Y^2+X-Y'
```

DIMS

Category Symbolic Matrix Manipulation

Affected by flag none

Input

1: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,n}\}$
$\ldots$
$\{Symb_{m,1}\ Symb_{m,2} \ldots Symb_{m,n}\}\}$

Output

2: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,n}\}$
$\ldots$
$\{Symb_{m,1}\ Symb_{m,2} \ldots Symb_{m,n}\}\}$
1: { r c}

Function Returns the dimensions of the list-matrix, checking matrix consistency. If a dimensional error is detected, an error messagge is issued.

Notes No check is made on object types. If you want to check it, use MSYMB?.

See Also CONFORM?, MATWRT, MSYMB?, SQUARE?

Applications INVRT, MATMENU, PRSYMB

EQUAL?

Category Symbolic Matrix Manipulation

Affected by flag none

Input

2:	{{SymbA$_{1,1}$ ... SymbA$_{1,n}$} ... {SymbA$_{m,1}$ ... SymbA$_{m,n}$}}	2:	{{SymbA$_{1,1}$ ... SymbA$_{1,n}$} ... {SymbA$_{m,1}$ ... SymbA$_{m,n}$}}
1:	{{SymbB$_{1,1}$ ... SymbB$_{1,n}$} ... {SymbB$_{m,n}$ ... SymbB$_{m,n}$}}	1:	{{SymbB$_{1,1}$ ... SymbB$_{1,q}$} ... {SymbB$_{p,1}$ ... SymbB$_{p,q}$}}

Output

4:	{{SymbA$_{1,1}$ ... SymbA$_{1,n}$} ... {SymbA$_{m,1}$ ... SymbA$_{m,n}$}}		
3:	{{SymbB$_{1,1}$ ... SymbB$_{1,n}$} ... {SymbB$_{m,1}$ ... SymbB$_{m,n}$}}	3:	{{SymbA$_{1,1}$ ... SymbA$_{1,n}$} ... {SymbA$_{m,1}$ ... SymbA$_{m,n}$}}
2:	{ m n }	2:	{{SymbB$_{1,1}$ ... SymbB$_{1,q}$} ... {SymbB$_{p,1}$ ... SymbB$_{p,q}$}}
1:	1	1:	0

Function Checks if both list-matrices have the same dimensions. If dimensions mismatch a 0 is returned.

Notes The underlying meta-object structure of the result unifies the boolean convention with the meta-object convention. The output is especially suitable for IFT or IFTE input

See Also CONFORM?, DIMS, MSYMB?, SQUARE?

Applications MATMENU

EXT→

Category Type Conversion

Affected by flag none

Input
1: obj

Output
1: addr

Function Returns the memory address where the object is stored. When the address is less than ph the object is stored in ROM.

Notes EXT→ is particularly useful to decipher External Objects. External is the raw representation of a system address. Making it a system binary, makes the thing a new ball game. Externals are explained in Appendix C. A list of Externals is given in Appendix D.

See Also →EXT, SYS→, →SYS.

Examples

1 EXT→ **[ENTER]**

1: <2A2C9h>

#2A2C9 SYSEVAL **[ENTER]**

1: 1

TRUE **[ENTER]**

1: External

EXT→ **[ENTER]**

1: <03A81h>

→EXT

Category Type Conversion

Affected by flag -5 to -10 (Binary Wordsize) and -11 to -12 (Binary Integer Base) only when the input number is a binary integer.

Input

1: addr

or

1: #addr

Output

1: Obj

Function Pushes on the stack a ROM address.

Notes If at the address specified begins a machine language routine, the stack display will show External on the first level. Unfortunately the HP-48 represents with 'External' meaningless address too, thus pay attention before using an External. When you supply an address at which an RPL object is stored, the Stack display will show you the correspondent text representation. A sequence of threads will be displayed as a stream of Externals.

Meaningful Externals are collected in the Appendix D.

See Also EXT→, →PRG.

Applications L→TH, SHRINK, TH→L.

Examples

1: #30794h

→EXT **[ENTER]**

1: External

[EVAL]

1: "HPHP48-x" x = revision letter (A,B,C,D or E)

1: #2B0F2h

→EXT **[ENTER]**

1: Long Real

FACTOR

Category Symbolic Matrix Manipulation

Affected by flag -3 (Symbolic Result)

Input

2: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,n}\}$
$\ldots$
$\{Symb_{m,1}\ Symb_{m,2} \ldots Symb_{m,n}\}\}$

1: Symb

Output

1: $\{\{Symb*Symb_{1,1}\ Symb*Symb_{1,2} \ldots Symb*Symb_{1,n}\}$
$\ldots$
$\{Symb*Symb_{m,1}\ Symb*Symb_{m,2} \ldots Symb*Symb_{m,n}\}\}$

Input

2: $\{\{SymbA_{1,1}\ SymbA_{1,2} \ldots SymbA_{1,n}\}$
$\ldots$
$\{SymbA_{m,1}\ SymbA_{m,2} \ldots SymbA_{m,n}\}\}$

1: $\{\{SymbB_{1,1}\ SymbB_{1,2} \ldots SymbB_{1,n}\}$
$\ldots$
$\{SymbBm1\ SymbB_{m,2} \ldots SymbB_{m,n}\}\}$

Output

1: $\{\{SymbA_{1,1}*SymbB_{1,1}\ SymbA_{1,2}*SymbB_{1,2} \ldots SymbA_{1,n}*SymbB_{1,n}\}$
$\ldots$
$\{SymbA_{m,1}*SymbB_{m,1}\ SymbA_{m,2}*SymbB_{m,2} \ldots SymbA_{m,n}*SymbB_{m,n}\}\}$

Function Multiplies all the elements of the list-matrix by the Symbolic value Symb or performs in-place multiplication between pairs of elements.

Notes A one-dimensional array must be written as one-row or one-column two-dimensional array. In order to avoid run-time errors the flag -3 must be clear, otherwise it will be necessary a numeric value for each symbol to carry out the calculation numerically.

The routine does not check for symbolic values. The reason is explained below:

The routine makes use of a hidden command which allows you to pass the operator unevaluated. Moreover the operator is not restricted only to symbolic object but can work on all pairs of elements for which that operation is meaningful.

See Also ADD, DETERM, MULT, SQUARE?, SUBT

Applications MATMENU

FALSE

Category	Type Conversion
Affected by flag	none
Input	
Output	1: External (FALSE)
Function	Pushes on the stack the system boolean FALSE.
Notes	System booleans are machine language routine addresses which merely return themselves when evaluated. The command →TorF turns a system boolean into a real boolean.
See Also	EXT→, TRUE, →TorF

FIND

Category String, List and program editing function.

Affected by flag none

Input

2:	str1	2:	prg	2:	list
1:	str2	1:	obj_1	1:	obj_1

Output

2: input_object
1: Found: occurences

Function Returns the total number of occurences of an object or substring within a composite object or string respectively.

See Also REPLACE

Examples

2: { "" 2 4 { 2 "" } }
1: ""

FIND **[ENTER]**

2: { "" 2 4 { 2 "" } }
1: Found: 2

2: "ABCDABCDEFABC"
1: "ABC"

FIND **[ENTER]**

2: "ABCDABCDEFABC"
1: Found: 3

1: « → n « n ROLL DROP »»

{ ROLL } OBJ→ DROP FIND **[ENTER]**

2: « → n « n ROLL DROP »»
1: Found: 1

IDNT

Category	Symbolic Matrix Manipulation
Affected by flag	none
Input	1: n
Output	1: {{1 0 ... 0} {0 1 ... 0} ... {0 0 ... 1}}
Function	Returns the identity list-matrix of order n.
See Also	CONSTMAT
Applications	MATMENU
Examples	1: 3 IDNT **[ENTER]** 1: {{ 1 0 0 } { 0 1 0 } { 0 0 1 }}

KEYWAIT

Category Input/Output

Affected by flag none

Input

Output

2: <KEYh>
1: <SHIFTh>

Function

Waits indefinitely for a keypress. The **[ON]** key is trapped like any other key.

Notes

The Keyboard is numbered, starting from the upper left corner down to the lower right corner in row major order. The key associated to A is numbered <1h> and the last is <2Fh> (49d). KEYWAIT detects shifted keys so that [ALPHA], **[Gold]** and **[Blue]** cannot be trapped singularly. The shift status is so encoded:

<1h>	no SHIFT	This encoding system requires
<2h>	**[Gold]**	fewer processing than that required
<3h>	**[Blue]**	to dispatch a key trapped with 0
<4h>	[alfa]	WAIT. Moreover the **[ON]** key is
<5h>	[alfa] **[Gold]**	trapped like any other key.
<6h>	[alfa] **[Blue]**	

The KEYWAIT internal routine is the most simple application of what HP calls Parameterized Outer Loop, also known as ParOuterLoop. This routine is the core of any interactive built-in application thanks to its flexibility. The GRAPHics editor uses the same basic routine as KEYWAIT ! ParOuterLoop is well explained in the documentation provided by HP as HP-48 RPL compiler Doc, you can easily get as EduCALC Goodies disc #4.

JOINR

Category	Graphics
Affected by flag	none
Input	2: $Grob_{m \times h}$ 1: $Grob_{p \times h}$
Output	1: $Grob_{(p+m) \times h}$
Function	Appends the grob on the first level to the right side of the grob on the second level.
Notes	Grobs must have the same height. The result is placed in a new grob.
See Also	JOINUP, RPT
Applications	→FONT, PROBJ

JOINUP

Category	Graphics
Affected by flag	none
Input	2: $Grob_{w \times h}$ 1: $Grob_{w \times l}$
Output	1: $Grob_{w \times (h+l)}$
Function	Appends the grob on the first level to the top side of the grob on the second level.
Notes	Grobs must have the same width. The result is placed in a new grob.
See Also	JOINR

L2M

Category List and Stack Manipulation

Affected by flag none

Input

N+1: "MARK'
N: obj_1
...: ...
2: obj_{n-1}
1: obj_n

Output

1: $\{obj_1\ obj_2 \ldots obj_n\}$

Function

Collects all the objects between the mark and the TOS. The mark is always removed from the stack.

Notes

If no mark is there on the stack, L2M defaults to DEPTH →LIST. This command is useful to group together the output of a command of which we cannot know in advance the exact number of parameters.

With this command you can set the user free to decide what parameter to use and what to discard without annoying him with verbose questions.

'MARK is a private mark. Hidden functions make use of an alternate mark to avoid collision with user defined programs.

See Also L2M, MARK, META

Examples

```
3:      'MARK
2:      "Enter as many numbers
         as you want"
1:      { V }
```

```
« IFERR INPUT
  THEN
    KILL
  END
  OBJ→
  L2M OBJ→
  1 -
  « MAX »
  RPT
  "The greatest is "
  SWAP +
  7 DISP 7 FREEZE
»
```

LINES→

Category	String and Meta-object Manipulation
Affected by flag	none
Input	1: str
Output	N+1: str_1 ....: ... 2: str_n 1: n
Function	Splits the string into several lines breaking at linefeeds.
Notes	Linefeeds are used by the 48 as newline characters in the editor. Moreover they are translated to the sequence CR LF during the transmission to a printer when the translation parameter in the global variable IOPAR has a value greater than 0. LINES→ removes the linefeeds, but blows up on the stack the string in several chunks. However the routine ignores linefeeds falling between double quotes.
See Also	→LINES.

→LINES

Category	String and Meta-object Manipulation
Affected by flag	none
Input	N : str_1 obj_1 ... : ... ... 2: str_n obj_n 1: n n
Output	1: str
Function	Joins objects by means of a linefeed.
Notes	It is the inverse of LINES→.
See Also	LINES→, SPLIT
Applications	PROBJ

LOC

Category String Manipulation

Affected by flag none

Input

3: str_1
2: str_2
1: pos

Output

1: pos

Function Seeks str_2 in str_1 starting from position pos. If no match is found it returns 0, otherwise the absolute location of the match.

Notes LOC is an extension of POS.

See Also MEMBER, SPAN

Examples

3: "abcdabcdabcd"
2: "abc"
1: 2

LOC **[ENTER]**

1: 5

LOP1

Category List Manipulation

Affected by flag none

Input

2: $\{obj_1\ obj_2 \ldots obj_n\}$
1: $\{cmd_1\ cmd_2 \ldots cmd_k\}$

Output

1: $\{obj_1\ obj_2 \ldots obj_n\}$

Function Given an operand string in level 2 and an operator string in level 1, applies the operator to each element of the operand-list and puts the result in a new list.

Notes The operator list must return one and only one result at a time. If the operator returns more than one object as result, use LOPN. This command has got aspects in common with the induction postulate:

1) define a procedure working on a single object
2) proof if it works on the first element.
3) apply to all elements.

See Also LOPN, LVOP, METOP

Examples

2: { 2 3 4 }
1: { INV →Q }

LOP1 **[ENTER]**

1: { '1/2' '1/3' '1/4' }

VARS **[ENTER]**

1: { PROG1 PROG2 PROG3 }

{ DUP BYTES NIP SWAP →TAG } LOP1 **[ENTER]**

1: { :PROG1:307 :PROG2:8604.5 :PROG3:1233 }

LOPN

Category List Manipulation

Affected by flag none

Input

2: $\{obj_1\ obj_2 \ldots obj_n\}$
1: $\{\ prg_1\ prg_2 \ldots prg_k\}$

Output

1: $\{\ \{obj_{1,1} \ldots obj_{1,p}\}$
...
$\{\ obj_{n,1} \ldots obj_{n,q}\}$

Function Given an operand list in level 2 and an operator list in level 1, applies the operators to each element of the operand-list, collects the results in a list and puts it in the result list.

Notes Each operator must take as argument a list. The output of the first operator is used as input for the second and so on.

See Also LOP1, LVOP, METOP

Examples VARS **[ENTER]**

1: { PROG1 PROG2 PROG3 }

{ BYTES } LOPN **[ENTER]**

1: {{ #1AE1 307 }
{#11D1 8718 }
{#113D 214.5}}

LVOP

Category List Manipulation

Affected by flag none

Input

3: { obj_1 obj_2 ... obj_n }
2: {obj_1 obj_2 ... obj_p}
1: { cmd_1 cmd_2 ... cmd_k}

Output

1: {obj_1 obj_2 ... $obj_{min(n,p)}$}

Function Applies each operator to the pairs of elements taken from the operand-list in levels 2 and 3.

Notes When the operand-lists have different size, exceeding objects are ignored. If you need to perform a calculation based on the current value of the counter, use the identifier 'idx' as counter. It will be replaced by the actual value.

See Also LOP1, LOPN, METOP

Examples

```
3:          { 1 2 3 }
2:          { 10 20 }
1:          { + }
```

LVOP **[ENTER]**

```
1:          { 11 22 }
```

```
3:          { 3 5 }
2:          { 1 1 }
1:          { SWAP / }
```

LVOP **[ENTER]**

```
1:          { 0.333333333333 0.2 }
```

```
3:          { 1 "" (2,0) 30 5}
2:          { 0 ""  X  30 7}
1:          { SAME { idx } IFT }
```

LVOP **[ENTER]**

```
1:          { 2 4 }
```

MARK

Category	Meta-object Manipulation and Stack Handling
Affected by flag	none
Input	
Output	1: "MARK'
Function	Puts the private mark on the stack. A mark delimits an unbound meta-object.
Notes	'MARK is an unresolved global name. Do not store any object in it.
See Also	C2M, L2M

MATWRT

Category Symbolic MatrixWriter

Affected by flag -15 through -18 and -45 through -50, -51

Input

1: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,n}\}$
...
$\{Symb_{m,1}\ Symb_{m,2} \ldots Symb_{m,n}\}\}$

Output

1: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,q}\}$
...
$\{Symb_{p,1}\ Symb_{p,2} \ldots Symb_{p,q}\}\}$

Function Allows interactive editing of a symbolic list-matrix. Details about the editor under CSTMENU.

Notes MATWRT is the sole visible command of library 822 (Symbolic MatrixWriter). Hidden commands of Library 822 are not supported and cannot be used for software development. Future SmartROM extension will use this Library number.

See Also CSTMENU

Applications CALENDAR

MEMBER

Category String Manipulation

Affected by flag none

Input

3: str_1
2: str_2
1: pos

Output

1: pos

Function

Returns the absolute position of the first character in Str_1 comprised in Str_2, starting from position pos. If no match is found, returns 0.

Notes

MEMBER is useful to skip text given a particular set of characters, typically punctuation characters or delimiters.

Frequently used string-constants has been stored in ROM to save memory:

DIGIT$ = "0123456789"
alfaLOW$ = "ABCDEFGH...XYZ"
alfaUPP$ = "abcdefgh...xyz"

They are accessible in the fifth page of menu $&L in CST.

Try also 821 250 →Xlib **[ENTER]** **[EVAL]** and 821 251 →Xlib **[ENTER]** **[EVAL]**

MEMBER and SPAN are useful to check input from the user, for the presence or absence of certain characters. Typically they are used to implement parser routines in conjunction with SPLIT, SUB and other String manipulation functions. Appendix H lists hidden string functions.

See Also SPAN, SPLIT

Examples

3: "123456789A12DEF"
2: "ABCDE"
1: 11

MEMBER **[ENTER]**

1: 13

META

Category	Meta-object Manipulation
Affected by flag	none
Input	1: n
Output	N+1: 1 ...: ... 3: n-1 2: n 1: n
Function	Creates a real meta-object in increasing order.
Notes	META is useful to create index arrays in conjunction with →ARRY.
See Also	METOP, NDUPN, SRT, STRD
Applications	CALENDAR

METOP

Category Meta-object Manipulation

Affected by flag depends on the operators passed for evaluation

Input

N+1:	obj_1
...:	...
3:	obj_n
2:	n
1:	list

Output

N+1:	obj_1
...:	...
2:	obj_n
1:	n

Function Applies a sequence of commands evaluating to a single result to each object of the meta-object. The final result is a meta-object of the same size as of the original one.

Notes The list in level 1 must contain a sequence of commands whose result is a single object. This convention, in practice, does not restrict the usage of METOP. When you use up an object doing some operation, you can refill the empty with a boolean or a dummy object.

See Also LOP1, LOPN, LVOP

Applications L→TH, TH→L

Examples

7:	#45h
6:	37
5:	<22h>
4:	12
3:	#11h
2:	5
1:	{ →SYS }

METOP **[ENTER]**

6:	<45h>
5:	<25h>
4:	<22h>
3:	<Ch>
2:	<11h>
1:	5

MGET

Category Symbolic Matrix Manipulation

Affected by flag none

Input

3: $\{\{Symb_{1,1} \ldots Symb_{1,k} \ldots Symb_{1,n}\}$
$\ldots$
$\{Symb_{m,1} \ldots Symb_{i,k} \ldots Symb_{m,n}\}$
$\ldots$
$\{Symb_{m,1} \ldots Symb_{m,k} \ldots Symb_{m,n}\}\}$

2: i

1: k

Output

1: $Symb_{i,k}$

Function Extracts an element from a list-matrix.

Notes No check is made on the type of the objects contained in the list, nor on the consistency of the structure of the matrix. This feature lets you extract elements from two dimensional lists of arbitrary structure. If the pointee is missing an error is issued.

See Also MPUT

Applications MATMENU, PRSYMB

MOVE

Category Meta-object Manipulation

Affected by flag none

Input

N+4:	obj_1
....:	...
....:	obj_{from}
....:	...
....:	obj_{to}
....:	...
....:	obj_{above}
....:	...
5:	obj_n
4:	n
3:	from
2:	to
1:	above

Output

N+1:	obj_1
....:	...
....:	obj_{from-1}
....:	...
....:	obj_{to+1}
....:	...
....:	$obj_{above-1}$
....:	obj_{from}
....:	...
....:	obj_{to}
....:	obj_{above}
....:	...
....:	obj_n
1:	n

Function Shifts the block of objects delimited by from and to above objectabove.

Notes Top of stack contains the counter while level 2 contains the last object of the meta-object (as if it were the last element of a list). The first element of the meta-object lies on level n+1(after the execution of the command).

Input parameters from and to can be given in any order. If you specify for above a value greater than n, the section will shift after the tail of the meta-object.Despite of COPY, MOVE does not allow a value for destination between from and to

See Also DELETE, COPY

Examples Follow on the next page.

Examples

6: "I stay here"
5: "I get moved"
4: "me too"
3: "I'll go up"
2: "End"
1: 5

2 3 5 MOVE **[ENTER]**

6: "I stay here"
5: "I'll go up"
4: "I get moved"
3: "me too"
2: "End"
1: 5

6: "I'll stay here"
5: "I get moved"
4: "me too"
3: "I'll go up"
2: "End"
1: 5

2 3 6 MOVE **[ENTER]**

6: "I stay here"
5: "I'll go up"
4: "End"
3: "I get moved"
2: "me too"
1: 5

MPUT

Category Symbolic Matrix Manipulation

Affected by flag none

Input

4: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,n}\}$
$\ldots$
$\{Symb_{m,1}\ Symb_{m,2} \ldots Symb_{m,n}\}\}$

3: i

2: k

1: Symb

Output

1: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,k} \ldots Symb_{1,n}\}$
$\ldots$
$\{Symb_{i,1}\ Symb_{i,2} \ldots Symb \ldots Symb_{i,n}\}$
$\ldots$
$\{Symb_{m,1}\ Symb_{m,2} \ldots Symb_{m,k} \ldots Symb_{m,n}\}\}$

Function Replaces the value contained at location (i,k) with Symb.

See Also MGET, MSYMB?

Applications MATMENU, PRSYMB

MREV

Category Meta-object Manipulation

Affected by flag none

Input

N+1:	obj_1
....:	...
2:	obj_n
1:	n

Output

N+1:	obj_n
....:	...
2:	obj_1
1:	n

Function Reverses the order of the objects in the meta-object.

See Also SRTD

Examples

1: [1 2 3 4 5 6]

OBJ→ OBJ→ DROP MREV →ARRY **[ENTER]**

1: [6 5 4 3 2 1]

MSBIT

Category Type Conversion

Affected by flag -5 through -10 for binary integers only

Input 1: n 1: <n> 1: #n

Output 1: msbit

Function Returns the position of the most significant bit in the mantissa of the input number.

Notes Real numbers are automatically rounded to integers before the operation.

MSBIT returns a value between 0 and 20. 0 means no bit set.

The value returned complies with the following definition:

MSBIT=INT(LOG2(n))+1 for n#0.

MSBIT=0 for n=0.

Examples

1: h

MSBIT **[ENTER]**

1: 3

MSYMB?

Category Symbolic Matrix Manipulation

Affected by flag none

Input

1: {{$Symb_{1,1}$ $Symb_{1,2}$ ... $Symb_{1,n}$}
...
{$Symb_{m,1}$ $Symb_{m,2}$ ... $Symb_{m,n}$ }}

Output

2: {{$Symb_{1,1}$ $Symb_{1,2}$ ... $Symb_{1,n}$ }
...
{$Symb_{m,1}$ $Symb_{m,2}$ ... $Symb_{m,n}$ }}
1: 1

or

m*n+2: $Symb_{1,1}$
m*n+1: $Symb_{1,2}$
... : ...
... : Syntax: Obj
... : ...
3: $Symb_{m,n}$
2: { 1 3 }
1: 0

Function Checks the contents of the list-matrix. Any object whose type is not a Real, Complex, Unit, Symbolic or Global is tagged with the string "Syntax" and the list-matrix is decomposed on the stack.

Notes MSYMB? does not check dimensions. To check dimensions use DIMS.

See Also CONFORM?, DIMS, EQUAL?, SQUARE?

MULT

Category Symbolic Matrix Manipulation

Affected by flag -3 (Symbolic Result)

Input

2: {{$SymbA_{1,1}$ $SymbA_{1,2}$ ... $SymbA_{1,n}$}
...
{$SymbA_{m,1}$ $SymbA_{m,2}$ ... $SymbA_{m,n}$}}

1: {{$SymbB_{1,1}$ $SymbB_{1,2}$ ... $SymbB_{1,p}$}
...
{$SymbB_{n,1}$ $SymbB_{n,2}$ ... $SymbB_{n,p}$}}

Output

1: {{$Symb_{1,1}$ $Symb_{1,2}$ ... $Symb_{1,p}$}
...
{$Symb_{m,p}$ $Symb_{m,p}$ ... $Symb_{m,p}$}}

Function Performs row by columns multiplication between symbolic arrays.

Notes List-matrices must have compatible dimensions, that is if the first array is a (m,n) the second must be (n,p). We called this special property conformability (see under CONFORM?). When the aforementioned condition is violated, an error is issued. If the flag -3 is set, run-time erros may happen if an identifier cannot be resolved to a numeric value. In-place multiplication is performed by FACTOR. The last example below shows you the difference between row by column and in-place multiplication.

See Also CONFORM?, FACTOR

Applications MATMENU, PRSYMB

Examples

```
2:  {{ 1 2 }{ 'X' 'X-1' }}
1:  {{ '-X' -1 }}
```

MULT **[ENTER]**

```
1:  {{ '-X-2' }
     { 'X*-X-(-1+X)' }}
```

```
2:  {{ 1 2 }{ 6 7 }}
1:  {{ 2 1 }{ 3 4 }}
```

MULT **[ENTER]**

```
1:  {{ 8 9 } { 33 34 }}
```

```
{{ 1 2 }{ 6 7 }}
{{ 2 1 }{ 3 4 }}
```

FACTOR **[ENTER]**

```
{{ 2 2 }{ 18 28 }}
```

NDUPN

Category Meta-object Manipulation

Affected by flag none

Input

```
2:      Obj
1:      n
```

Output

```
N+1:    Obj
...:    ...
2:      Obj
1:      n
```

Function Creates a meta-object by duplicating a given object.

Notes If n=0 NDUPN creates a null meta-object.

See Also META

Examples

```
1:      { hello }
```

4 NDUPN **[ENTER]**

```
5:      { hello }
4:      { hello }
3:      { hello }
2:      { hello }
1:      4
```

```
1:      { hello }
```

0 NDUPN

```
1:      0
```

NIP

Category	Stack Manipulation
Affected by flag	none
Input	2: Obj_A 1: Obj_B
Output	1: Obj_B
Function	Removes from the stack the object on the second level.
See Also	AAB, BAA, BAB, BBA, BCAC, BCDA, CAB, CBA

NULL

Category Type Conversion

Affected by flag none

Input

1: obj

Output

1: obj (null)

Function

Replaces the input object with the null object of the same type respect to the addition operation.

Only the objects listed below have a correspondent null object.

Type	Null element
0	0
1	(0,0)
2	""
3	[0...0] or [[0...0]...[0...0]]
4	[(0,0)...(0,0)] or [[(0,0)...(0,0)]...[(0,0)...(0,0)]]
5	{}
6	0
7	0
9	0
10	#0h
11	Blank $_{w \times h}$
12	inherits from the ancestor if defined.
13	0_unit
20	<0h>

Notes

A Tagged object inherits ancestor's type.

Polymorphism is a property of RPL language (at user level). It allows you to design object-independent algorithms. Of course some operations make sense only with certain entities, but setting the algorithm free from object slavery, you will save time later, when you need to recycle the routine.

The NULL command lets you design recursive algorithms or loop structures independently from the input object type. Typically such algorithms need some initialization code in order to start a chain calculation. The + (plus) Command is the most flexible operator built in the 48. The NULL command lets you initialize every routine based on concatenation or addition without knowing in advance the object type.

See Also RPT

Examples

:tag:{ 1 2 3 } NULL **[ENTER]**

```
1:      { }
```

```
1:      [[ 0 1  5]
         [ 4 3 -2]]
```

NULL **[ENTER]**

```
1:      [[ 0 0 0 ]
         [ 0 0 0 ]]
```

PARSE

Category String Manipulation

Affected by flag -5 through -10 (wordsize), -15 and -16(coordinate system), -17 and -18 (angle mode)

Input

1: str

Output

		4:	str
		3	<last>
2:	prg	2:	"characters"
1:	External (TRUE)	1:	External (FALSE)

Function Performs the parsing of the input string. If no syntax error is detected an object containing the executable code is returned along with the system boolean TRUE. Otherwise the original string, the absolute position of the last character scanned and the text containing the syntax error are returned along with the system boolean FALSE.

Notes By extending the system parser, it is possible to handle unsupported object types like system binaries. The example given below shows a possible technique, you could enhance at your will.

on-the-fly parser handling system binaries:

« 0 → current	initializes replacement counter
« {} '$SUBST' STO	initializes temporary storage area
DO PARSE →TorF	parses input string
IF NOT THEN	if error then
NIP	take apart the string
SPLIT OVER DUP	
'$SUBST' STO+	and save it.
CAB + SWAP	take the rest of the string
"$sub" 'current' INCR +	give a name to the substring
REPLACE DROP	replace all the occurences of the string with the identifier
+	reconstruct the string for parsing.
0	prepare to parse again
ELSE 1	exit parsing
END	
UNTIL	

```
END                          loopback if error during parse
 1 →PRG                      make it a program
IF current THEN              if any replacement
$SUBST
OBJ→ DUP 2 +                 retrieve strings and prepare for
                             loop
ROLL 1 ROT
FOR i
"$sub" i + OBJ→              build indentifier
ROT 1 SPLIT                  parse unrecognized text
ROT  DROP NIP "#"
SWAP + EndOfString SPLIT
DROP DUP
IF "" SAME
THEN DROP
ELSE +
END
OBJ→ →SYS
REPLACE DROP                 convert into a sysbin and
                             replace all the occurrences.
EXT →
END » '$SUBST'
PURGE EVAL »
'Parse' STO
```

Try it on this string

```
"« (0,0) 3 0 360 60 <1h> <2h> 821 247 →Xlib »"
1:      « (0,0) 3 0 360
          60 <1h> <2h>
          821 247 →Xlib »
EVAL [GRAPH]
```

PKMETA

Category Meta-object Manipulation

Affected by flag -56 Beep

Input

N+5:	obj_1
....:	...
6:	obj_n
5:	n
4:	begin
3:	current
2:	lines
1:	row

Output

[ENTER]

N+5:	obj_1
....:	...
6 :	obj_n
5:	n
4:	begin
3:	current
2:	objcurrent+begin
1:	1

[ON]

N+4:	obj_1
....:	...
5:	obj_n
4:	n
3:	begin
2:	current
1:	0

Function Shows a catalog of a specified number of lines beginning on a specified display line. The selection of the object is interactive and mantains the functionalities of the built-in catalog.

Notes Command parameters are so defined :

obj_1
...
obj_n
n

input meta-object.

begin

index of the first object on which beginning the page minus one (from 0 to n-lines).

current

current element within the page (from 1 to lines).

lines

height of the page in lines (from 1 to 8-row). row starting row of the display (from 1 to 7). When using a value less than 2, FREEZE may be required.

Values exceeding the limits are rejected with an error.

Selecting an object means moving the pointer to the line containing the object and press **[ENTER]**. Once you press **[ENTER]** the catalog is exited and the selected object is duplicated as shown in the stack diagram. The selection may be aborted by pressing **[ON]**. In this case only the last pointer position is returned. Arrow keys perform

pointer movements in the same way the built-in catalog allows.

PKMETA is very flexible because each action associated to a key may be redefined. Each defined key must be associated to a program stored in user memory. The following table summarizes the PKMETA auxiliary programs and data structures and the default keys along with their actions:

Global Name	Keycode	Action
KEYS	n/a	Mantains the list of defined keys. Each element of the list is a list containing two system binaries. The first number represents the absolute key number h to Dh, the second represents the shift plane h to h. This encoding system matchs KEYWAIT format.
ACTIONS	n/a	Mantains the list of the actions associated to the keys. The list must always have at least one element. The first element is a name of the program that must be called when an undefined key is pressed. By default this name is BADKEY. The second element corresponds to the first keycode stored in KEYS and so on. ACTIONS must always contain a number of actions equal to SIZE(KEYS)+1. Otherwise a special error code will be issued.
BADKEY	n/a	The action taken when an undefined key has been pressed.
ATTN	<2Dh> <1h>	The action associated to the pressing of **[ON]**. By default it aborts the selection.
DOWNARR	<11h> <1h>	The action associated to the pressing of **[↓]**. It moves the pointer downwards, eventually scrolling up the page by one line.
UPARR	<0Bh> <1h>	The action associated to the pressing of **[↑]**. It moves the pointer upwards, eventually scrolling down the page by one line.
ENTER	<19h> <1h>	The action associated to the pressing of **[ENTER]**. By default it confirms the selection and exits the catalog.
PGUP	<0Bh> <2h>	The action associated to the pressing of **[Gold][↑]** . It displays the previous page.
PGDOWN	<11h> <2h>	The action associated to the pressing of **[Gold] [↓]**. it displays the next page.
TOP	<0Bh > <3h>	The action associated to the pressing of **[Blue] [↑]**. Moves the pointer to the top of catalog.
BOTTOM	<11h> <3h>	The action associated to the pressing of **[Blue] [↓]**. Moves the pointer to the bottom of the catalog.
ENHANCE	n/a	The routine that displays the current line and the pointer.

NOR	n/a	The routine that cancels the pointer by the current line.

All the programs described above can be modified at your will. There are 5 parameters stored in temporary variables which contain the information you need to take some action. They represent the current value of the input parameters passed on the stack. Please note that this values are stored as *system binaries* and not as real numbers.

n	counter	Keeps stored the total number of elements of the meta-object (the counter)
o	begin page	This is the offset to the first line (element) which begins the page.
c	current element	This is the current element within the page.
r	row	This is the row of the display on which the page is anchored.
h	lines	This is the total number of lines per page.

These variables can be accessed by name or by their order in the temporary variables chain. If you know the entry points to recall and store temporary identifiers by their creation order, you can use them freely. The variables have been created in the order shown in the table above, that is n is the fifth of the chain and h is the first. To recall h use entry point #613B6h. However the safest way is to recall and store them by name.

Remember to change the current values according to underlying data structure. When PKMETA is running, on the stack there is only the body of the meta-object without the counter.

You can add or modify or change name simply modifying the ACTIONS list and updating, if necessary the KEYS list. All the customization of the command is with you.

PRG→

Category Program Editing, Meta-object Manipulation and Type Conversion

Affected by flag none

Input

1:	prg

Output

N+1:	ext_1
...:	...
2:	ext_n
1:	n

Function Splits a program in its meta-object form.

Notes A program is a collection of objects and pointers like a list or an algebraic expression. The main difference between programs and other composite object lies in its direct execution capability opposite to the indirect execution capabilities of lists and algebraics. Direct execution means that once the prolog of the program is executed, it starts executing objects within the program, while lists and algebraics merely push themselves on the stack. Once they are on the stack these objects can be executed via EVAL. Using PRG→ you will be able to modify in whatever manner you want a compiled program, deleting, moving, changing the objects it contains. The possibility to modify compiled programs directly on the stack makes much faster editing session of large programs, typically when you need to swap objects or make little changes in the source. Nevertheless PRG→ opens a wide range of applications dealing with program editing and in fact the SmartROM uses heavily this kind of commands. We suggest you try to edit a program via Interactive Stack.

May be you ramain quite surprised after expanding a User RPL program on the stack. In fact the built-in parser often adds hidden threads to perform safely dangerous operations like pushing an identifier on the stack unevaluated. Pay attention not to delete these hidden threads ! Moreover, as you will see, programming structures collect the commands between delimiters in a program object. To expand this kind of program, you need to move the object on the first level of the stack and call PRG→ again, then after editing it, you must recontruct the program with →PRG and move back the program to the original position.

A nice thing about PRG→ is that it can split also built-in functions and commands as SIN or STO. These commands appear, as any other internal program, as a stream of pointers to machine language routines and objects whose interpretation is impossible without commands like EXT→. If you want to understand the difference between so-called functions and commands, read the Notes under command

See Also EXT→, →EXT, →PRG

Applications L→TH, SHRINK, TH→L, UPTRIM

Examples

```
« IF 0 >
  THEN DROP SWAP
  END »
```

PRG→ **[ENTER]**

```
9:    «
8:    IF
7:    0
6:    >
5:    THEN
4:    DROP SWAP
3:    END
2:    »
1:    8
```

→PRG

Category	Meta-object Manipulation, Program Editing and Type Conversion
Affected by flag	none
Input	N+1: ext_1 ...: ... 2: ext_n 1: n
Output	1: prg
Function	Builds up a program object from a meta-object.
Notes	See under PRG→.
See Also	EXT→, →EXT, PRG→
Applications	L→TH, SHRINK, TH→L, UPTRIM

→R

Category	Type Conversion
Affected by flag	-5 through -10 when the input is a binary integer
Input	1: <nh> 1: #n 1: Char
Output	1: n
Function	Converts the input number into a real.
Notes	→R accepts also real numbers as input. This feature sets you free to use →R also when the object type should not require any conversion.
See Also	→B, →Char, EXT→, →EXT, →SYS

RDOWN

Category Stack Manipulation

Affected by flag none

Input

N+2:	obj_1
...:	...
T+2:	obj_t
...:	...
3:	obj_n
2:	n
1:	t

Output

N:	obj_t
...:	...
N-T+1:	obj_n
N-T:	obj_1
...:	...
1:	obj_{t-1}

Function Rolls down n objects t times.

Notes The command is smart enough to choose the best roll direction (upwards or downwards). Rolling down 100 objects 99 times is a good exercise of aerobyc dance for your 48, but it is not that kind of exercise we really need to do. We had better to roll up 100 objects one time !

See Also RUP, XLVLS

RDROP

Category Stack Manipulation

Affected by flag none

Input

....:	...
U+2:	liv_u
....:	...
P+2:	liv_p
....:	...
3:	liv_1
2:	p
1:	u

Output

....:	...
U :	liv_{u+1}
P-1:	liv_{p-1}
....:	...
1:	liv_1

Function Deletes the segment of the stack delimited by levels p and u.

Notes p and u can be given in any order.

See Also RDUP, SHIFT, XLVLS

Examples

```
5:    "first"
4:    "to get rid"
3:    "to get rid"
2:    "second-last"
1:    "last"
```

4 3 RDROP **[ENTER]**

```
3:    "first"
2:    "second-last"
1:    "last"
```

RDUP

Category Stack Manipulation

Affected by flag none

Input

...:	...
P+3:	liv_p
...:	...
U+3:	liv_u
...:	...
D+3:	liv_d
...:	...
4:	liv_1
3:	p
2:	u
1:	d

Output

...:	...
...:	liv_p
...:	...
...:	liv_u
...:	...
...:	liv_{d-1}
...:	liv_p
...:	...
D+1:	liv_u
D :	liv_d
...:	...
1:	liv_1

Function Copies the segment of the stack delimited by p and u above level d.

See Also RDROP, SHIFT, XLVLS

Examples

5:	"first"
4:	"get copied"
3:	"get copied"
2:	"second-last"
1:	"last"

4 3 1 RDUP **[ENTER]**

7:	"first"
6:	"get copied"
5:	"get copied"
4:	"second-last"
3:	"get copied"
2:	"get copied"
1:	"last"

REPLACE

Category String Function, List Manipulation and Program Editing

Affected by flag none

Input

3:	str_1	3:	Prg	3:	List
2:	str_2	2:	obj_1	2:	obj_1
1:	str_3	1:	obj_2	1:	obj_2

Output

2:	str	2:	Prg	2:	List
1:	Replaced:n	1:	Replaced:n	1:	Replaced:n

Function Substitutes all the occurences of the search-key with the object or string given and returns the total number of replacements.

Notes The replacements are limited to objects stored in user memory. Because some internal routines are recursive or reentrant, the search level is limited to threads stored in RAM. This preserves from endless loops. Internal routines of the SmartROM are able to perform selected substitutions at arbitrary depth within threads. Refer to the Hidden Commands Reference for more information on this topic.

See Also FIND

Examples

```
3:  « 4 ROLL SWAP DROP { OVER 4 ROLL } »
2:  ROLL
1:  ROLLD
```

REPLACE **[ENTER]**

```
2:  « 4 ROLLD SWAP DROP { OVER 4 ROLLD } »
1:  Replaced: 2
```

```
3:  "ABC ABC ABC"
2:  "ABC"
1:  "HELLO"
```

REPLACE **[ENTER]**

```
2:  "HELLO HELLO HELLO"
1:  Replaced: 3
```

REV

Category String Manipulation

Affected by flag -5 to -10 (binary integers only)

Input

1:	str	1:	{obj_1 obj_2 ... obj_n}	1:	#abcde

Output

1:	str	1:	{obj_n obj_{n-1} ... obj_1}	1:	#edcba

Function Reverses the order of the characters for strings, the order of the objects for lists and the order of the digits for binary integers.

Notes Binary integers are reversed according with their actual size. User binary integers may be no longer than 16 nibbles (in hex mode). However the 48 can handle binary integers of arbitrary size. For example, when you apply BYTES to an object, the binary checksum you get is always 4 nibbles long, no matter the current wordsize is. Of course the display shows it according to the wordsize, but its size remains 4 nibbles.

See Also MREV

Examples

1: "123456789A12DEF"

REV **[ENTER]**

1: "FED21A987654321"

1: { A B C }

REV **[ENTER]**

1: { C B A }

Supposing current wordsize of 64 bit.

1: #123456h

REV **[ENTER]**

1: #6543210000000000h

ROMV

Category	ROM Version
Affected by flag	none
Input	
Output	
Function	Shows information about the SmartROM.

ROWCOL

Category String Manipulation

Affected by flag none

Input

2: str
1: pos

Output

3: str
2: row
1: col

Function Computes the coordinate of the absolute position of a character in terms of rows and columns, by counting the linefeeds contained in the input string.

Notes The coordinates returned by ROWCOL can be used as parameters in the input list of command INPUT to place the cursor at a certain point within the editor. Typically this method is used to place the cursor on a particular occurence of a substring, previously found with LOC or POS.

Examples

"HELLO BOYS, THIS IS THE THIRD LINE OF TEXT" **[ENTER]**

DUP SIZE ROWCOL **[ENTER]**

3: "HELLO BOYS, THIS I..."
2: 3
1: 18

RPT

Category String Function, List Manipulation, Utility, Graphics

Affected by flag none

Input

2:	str	2:	n
1:	n	1:	str

or

2:	{ Obj_1... Obj_n}	2:	n
1:	n	1:	{ Obj_1... Obj_n}

or

2:	#b	2:	n	2:	$Grob_{w\ x\ h}$	2:	n
1:	n	1:	#b	1:	n	1:	$Grob_{w\ x\ h}$

or

2:	prg	2:	n	2:	Global	2:	n
1:	n	1:	prg	1:	n	1:	Global

Output

1:	str...str	1:	{ Obj_1... Obj_n... Obj_1... Obj_n}
1:	#bbbb..b	1:	$Grob_{(w*n)\ x\ h}$

or

N :	Obj
...:	...
1:	Obj

Function Chains data objects n times or executes n times a given procedure or identifier.

Notes RPT is one of the most flexible commands of the SmartROM. Thanks to its fast loop generator, it can link string and list quicker than any other command seen up to date. Try with a string of ten characters repeated 1000 times. You want believe to your eyes !

RPT is useful at most when you need to perform iterated operations without referencing counters. RPT does not mantain stack integrity, so that you can push or drop objects from the stack freely. If you have a DEMO program you want to iterate almost indefinitely, try this:

'DEMO' 1000000 RPT.

The loop normally cannot be interrupted. If you need to interrupt it press **[ON]** [C].

In the examples given below procedures are standard programs. Nevertheless you can push on the stack individual commands by doing so:

{ MAX } OBJ→ DROP

See Also METOP

Examples

```
2:      { 1 2 3 }
1:      "ABC"
```

2 { 3 RPT } METOP **[ENTER]**

```
3:      { 1 2 3 1 2 3 1 2 3 }
2:      "ABCABCABC"
1:      2
```

« RAND 10 * IP » 10 RPT

```
10:     3
9:      7
8:      5
7:      2
6:      7
5:      0
4:      8
3:      3
2:      6
1:      0
```

« MAX » 10 RPT **[ENTER]**

```
1:      8
```

Suppose you want to move a hundred variables from user memory to PORT 1 where you have a 128 K RAM:

1: {$Name_1$ $Name_2$ ... $Name_{100}$}

OBJ→ « DUP RCL BAA PURGE 1 →TAG STO »
RPT **[ENTER]**

RUP

Category Stack Manipulation

Affected by flag none

Input

N+2:	obj_1
...:	...
T+2:	obj_t
...:	...
3:	obj_n
2:	n
1:	t

Output

N:	obj_{t+1}
...:	...
T+1:	obj_n
T :	obj_1
...:	...
1:	obj_t

Function Rolls up n objects t times.

Notes The command is smart enough to choose the best roll direction (upwards or downwards). Rolling up 100 objects 99 times is silly. It is better roll down 1 time !

See Also RDOWN, XLVLS

SHIFT

Category Stack Manipulation

Affected by flag none

Input

...:	...
P+3:	liv_p
...:	...
U+3:	liv_u
...:	...
D+3:	liv_d
...:	...
4:	liv_1
3:	p
2:	u
1:	d

Output

...:	...
P :	liv_{p-1}
P-1:	liv_{u+1}
...:	...
...:	liv_{d-1}
...:	liv_p
...:	...
D+1:	liv_u
D :	liv_d
...	
1:	liv_1

Function Moves a segment of the stack comprised from level p to level u above level d.

Notes Stack-oriented commands require one parameter less than their meta-object-oriented counterparts. On the other hand meta-objects let you know exactly how many objects you shall deal with.

See Also RDROP, RDUP, XLVLS

Examples

5:	"I stay here"
4:	"I get moved"
3:	"me too"
2:	"I'll go up"
1:	"End"

4 3 1 SHIFT **[ENTER]**

5:	"I stay here"
4:	"I'll go up"
3:	"I get moved"
2:	"me too"
1:	"End"

SPAN

Category String Manipulation

Affected by flag none

Input

3: str_1
2: str_2
1: pos

Output

1: pos

Function

Returns the absolute position of the first character in Str_1 not comprised in Str_2, starting from position pos. If no match is found, returns 0.

Notes

A typical usage of SPAN is when checking for the presence of extraneous characters, especially when the input string comes from the user. Suppose the user must enter a numeric value without decimal point and Exponent. To check the string you can do so:

3: "758833"	This is the string given by the user.
2: "0123456789"	This is the string containing allowed characters
1: 1	Beginning position

SPAN **[ENTER]**

1: 0

Another frequent usage is when you need to skip blanks between words. In this case the test string must contain only a blank : " ". the position returned (if any) is that of next non-blank character. You could also ignore periods or any other punctuation by appending them to the test string.

" .,;"	lets you skip blanks, periods, commas and semicolons.

Frequently used string-constants has been stored in ROM to save memory:

```
DIGIT$  = "0123456789"
alfaLOW$ = "ABCDEFGH...XYZ"
alfaUPP$ = "abcdefgh...xyz"
```

They are accessible in the fifth page of menu $&L in CST.

Try also 821 250 ÄXlib **[ENTER] [EVAL]** and 821 251 ÄXlib **[ENTER] [EVAL]**

See Also MEMBER

Examples

```
3:              "123456789A12DEF"
2:              "1234567890"
1:              3
```

SPAN **[ENTER]**

```
1:              10
```

SPLIT

Category String Function and List Manipulation

Affected by flag none

Input

2:	str	2:	{ obj_1 obj_2 ... obj_p ... obj_n }
1:	p	1:	p

or

2:	p	2:	p
1:	str	1:	{ obj_1 obj_2 ... obj_p ... obj_n }

or

2:	str	2:	{ obj_1 obj_2 ... obj ... obj_n }
1:	str	1:	obj

Output

3:	str_1	3:	{obj_1 ... obj_{p-1}}
2:	str_2	2:	objp
1:	str_3	1:	{obj_{p+1} ... obj_n}

Function Splits the string or the list in three chunks:

Notes

3: Beginning chunk level
2: Middle chunk level
1: End Chunk

Empty string or lists are valid input objects.

If p is greater than the total size of the object, it is considered as SIZE(obj). If p is equal to 0 or the search-key is missing an error is issued.

Please note that the original object can be recontructed with two consecutive + (addition) operations.

Examples Follow on the next page.

Examples

```
2:      "123456789A12DEF"
1:      5

SPLIT [ENTER]

3:      "1234"
2:      "5"
1:      "6789A12DEF"

2:      "123456790"
1:      "456"

SPLIT [ENTER]

3:      "123"
2:      "456"
1:      "790"

2:      {1 2 3 4 5}
1:      6

SPLIT [ENTER]

3:      { 1 2 3 4 }
2:      5
1:      { }

2:      { 1 "abc" 'x/y'}
1:      "abc"

SPLIT [ENTER]

3:      { 1 }
2:      "abc"
1:      { 'x/y' }
```

SQUARE?

Category Symbolic Matrix Manipulation

Affected by flag none

Input

1: {{ $Symb_{1,1}$ ... $Symb_{1,n}$ }
...
{ $Symb_{n,1}$ ... $Symb_{n,n}$ }}

Output

3:	{{ $Symb_{1,1}$ ... $Symb_{1,n}$ } ... { $Symb_{n,1}$ ... $Symb_{n,n}$ }}		
2:	{ n n }	2:	{{ $Symb_{1,1}$ ... $Symb_{1,n}$ } ... { $Symb_{m,1}$ ... $Symb_{m,n}$ }}
1:	1	1:	0

Function Checks if the list-matrix is square. If yes it returns 1 along with matrix dimensions otherwise 0.

See Also DETERM, EQUAL?, CONFORM?

Applications INVRT, MATMENU, PRSYMB

Examples

1: {{ X Y Z }
{ 1 3 -2 }}

SQUARE? **[ENTER]**

2: {{ X Y Z } { 1 3 -2 }}
1: 0

SRDIFF

Category List Manipulation

Affected by flag none

Input

2: List
1: obj

Output

1: pos

Function

Returns the position of the first object different than Obj. If all objects are the same as Obj returns 0.

Notes

The implementation of a routine performing a test on all elements of a list is straightforward in internal RPL.

Pass 1	Create a test procedure taking two objects from the stack and returning a system boolean (See TRUE and FALSE).
Pass 2	Store it in a variable for easier reference.
Pass 3	Push on the stack the list and the object being tested
Pass 4	Push on the stack the name of the variable or directly the test procedure.
Pass 5	#64426 SYSEVAL **[ENTER]**

See Also SRGE, SRGT, SRLE, SRLT

Examples

2: { 1 1 1 1 1 2 3 }
1: 1

SRDIFF **[ENTER]**

1: 6

SRGE

Category List Manipulation

Affected by flag none

Input

2: { n_1 n_2 ... n_p }
1: n

Output

1: pos

Function Returns the position of the first real number greater or equal than n, otherwise returns 0.

Notes See under SRDIFF.

See Also SRDIFF, SRGT, SRLE, SRLT

Examples

2: { 1 1 1 1 1 3 6 }
1: 2

SRGE **[ENTER]**

1: 6

SRGT

Category List Manipulation

Affected by flag none

Input

2: $\{n_1\ n_2 \ldots n_p\}$
1: n

Output

1: pos

Function Returns the position of the first real number greater or equal than n, otherwise returns 0.

Notes See under SRDIFF.

See Also SRDIFF, SRGE, SRLE, SRLT

Examples

2: { 1 1 1 1 1 3 6 }
1: 3

SRGT **[ENTER]**

1: 7

SRLE

Category List Manipulation

Affected by flag none

Input

2: { n_1 n_2 ... n_p }
1: n

Output

1: pos

Function Returns the position of the first real number greater or equal than n, otherwise returns 0.

Notes See under SRDIFF.

See Also SRDIFF, SRGE, SRGT, SRLT

Examples

2: { 3 3 4 2 5 3 6 }
1: 2.9

SRLE **[ENTER]**

1: 4

SRLT

Category List Manipulation

Affected by flag none

Input

2: { n_1 n_2 ... n_p }
1: n

Output

1: pos

Function

Returns the position of the first real number greater or equal than n, otherwise returns 0.

Notes

See under SRDIFF.

See Also

SRDIFF, SRGE, SRGT, SRLE

Examples

2: { 3 4 10 2 1 3 6 }
1: 2

SRLT **[ENTER]**

1: 5

SRT

Category Meta-object Manipulation

Affected by flag none

Input

N+1:	obj_1
....:	...
2:	obj_n
1:	n

Output

N+1:	obj_1
....:	...
2:	obj_n
1:	n

Function Sorts the data in ascending order.

Notes Objects must be compatible with the < operator (less than). To this category of objects belong:

Global names
Real numbers
Binary integers
Strings
System Binaries
Tagged objects falling in one of the classes listed above

If you want to sort local names you need first to translate into global names, then use SRT and convert them back to Locals. To convert a Local name into a Global name back and forth use the following procedure:

#2464F SYSEVAL	Local to Global
#2465F SYSEVAL	Global to Local

To apply the translation to all the identifiers you can do the following:

N+2:	Local
....:	...
3 :	Local
2:	n
1:	{ #2464Fh SYSEVAL }

METOP **[ENTER]**

The inverse function needs only #2465Fh instead of #2464Fh.

Symbolic values are not allowed. The hidden code is able to sort any data for that a sort procedure has been defined. This means that you could sort any object given a sort criterion. Refer to the Hidden Commands Reference for further information.

See Also SRTD, MREV

Applications alfaORDER

Examples

```
9:      "JOHN"
8:      "MARY"
7:      "JIM"
6:      "STAN"
5:      "FRED"
4:      "paul"
3:      "LUISE"
2:      "HENRY"
1:      8
```

SRT **[ENTER]**

```
9:      "FRED"
8:      "HENRY"
7:      "JIM"
6:      "JOHN"
5:      "LUISE"
4:      "MARY"
3:      "STAN"
2:      "paul"
1:      8
```

SRTD

Category Meta-object Manipulation

Affected by flag none

Input

N+1:	obj_1
...:	...
2:	obj_n
1:	n

Output

N+1:	obj_n
...:	...
2:	obj_1
1:	n

Function Sorts data in descending order.

Notes Objects must be compatible with the > operator (greater than).

To this category of objects belong:

Global names
Real numbers
Binary integers
Strings
System Binaries
Tagged objects falling in one of the classes listed above

Further information under SRT.

See Also SRT

SUBT

Category Symbolic Matrix Manipulation

Affected by flag -3 (Symbolic Result)

Input

2: $\{\{SymbA_{1,1}\ SymbA_{1,2} \ldots SymbA_{1,n}\}$
$\ldots$
$\{SymbA_{m,1}\ SymbA_{m,2} \ldots SymbA_{m,n}\}\}$

1: $\{\{SymbB_{1,1}\ SymbB_{1,2} \ldots SymbB_{1,n}\}$
$\ldots$
$\{SymbB_{m,1}\ SymbB_{m,2} \ldots SymbB_{m,n}\}\}$

Output

1: $\{\{SymbA_{1,1}\text{-}SymbB_{1,1}\ SymbA_{1,2}\text{-}SymbB_{1,2} \ldots SymbA_{1,n}\text{-}SymbB_{1,n}\}$
$\ldots$
$\{SymbA_{m,1}\text{-}SymbB_{m,1}\ SymbA_{m,2}\text{-}SymbB_{m,2} \ldots SymbA_{m,n}\text{-}SymbB_{m,n}\}\}$

Function Subtracts the second matrix from the first. Matrices must have the same dimension.

See Also ADD, ADDCON, DIMS, EQUAL?, MULT, MSYMB?

Applications MATMENU, PRSYMB

SYMBMAT→

Category Symbolic Matrix Manipulation

Affected by flag none

Input

1: {{$Symb_{1,1}$ $Symb_{1,2}$ ... $Symb_{1,n}$ }
...
{$Symb_{m1}$ $Symb_{m2}$ ... $Symb_{m,n}$ }}

1: [[$N_{1,1}$ $N_{1,2}$ ... $N_{1,n}$]
...
[$N_{m,1}$ $N_{m,2}$ ... $N_{m,n}$]]

Output

N+1:	$Symb_1$	N+1:	$N_{1,1}$
....:	...	....:	...
2:	$Symb_{m,n}$	2 :	$N_{m,n}$
1:	{m n}	1 :	{m n}

Function Decomposes the list-matrix or the array on the stack.

Notes SYMBMAT→ is useful to convert ordinary numeric arrays into their symbolic counterpart. It handles numeric vectors as if they were arrays 1 x m.

Note that doing:

EVAL * you get a meta-object

See Also DIMS, MSYMB?, →SYMBMAT, TRNSP

Applications MATMENU, PRSYMB

Examples

```
1:      {{ 1 2 3 }
        { 'X' 'X-1' -1 }}
```

SYMBMAT→ **[ENTER]**

```
7:      1
6:      2
5:      3
4:      'X'
3:      'X-1'
2:      -1
1:      { 2 3 }
```

```
1:      [ 1 2 3 4 5 ]
```

SYMBMAT→ **[ENTER]**

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

→SYMBMAT

Category Symbolic Matrix Manipulation

Affected by flag none

Input

N+1: $Symb_1$
...:
3: $Symb_{n-1}$
2: $Symb_n$
1: {r c}

Output

1: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,c}\}$
...
$\{Symb_{r,1}\ Symb_{r,2} \ldots Symb_{r,c}\}\}$

Function Assembles the data on the stack in a symbolic matrix. If not enough objects are on the stack an error is issued.

Notes The command does not check the type of the objects. This feature lets you build list-matrices for arbitrary purposes.

See Also CONSTMAT, DIMS, MSYMB?, SYMBMAT→, TRNSP

Applications MATMENU, PRSYMB

Examples

7: 1
6: 2
5: 3
4: 'X'
3: 'X-1'
2: -1
1: { 2 3 }

→SYMBMAT **[ENTER]**

1: {{ 1 2 3 }
{ 'X' 'X-1' -1 }}

→SYS

Category	Type Conversion
Affected by flag	-5 through -10 bynary integer wordsize (binary integers only)
Input	1: n 1: #n 1: Char
Output	1: <nh>
Function	Converts an input number into a system binary.
Notes	→SYS accepts system binaries as well. This feature sets you free from checking in advance the type of the input object. System binaries are explained in detail in Appendix C.
See Also	→B, →Char, →EXT, →R

→TorF

Category	Type Conversion			
Affected by flag	none			
Input	1:	TRUE	1:	FALSE
Output	1:	1	1:	0
Function	Converts the system boolean value into a numeric boolean.			
See Also	FALSE, TRUE, EXT→, →EXT			

TRNSP

Category Symbolic Matrix Manipulation

Affected by flag none

Input

1: {{$Symb_{1,1}$ $Symb_{1,2}$ ... $Symb_{1,n}$}
...
{$Symb_{m,1}$ $Symb_{m,2}$ ... $Symb_{m,n}$}}

Output

1: {{$Symb_{1,1}$ $Symb_{2,1}$ ... $Symb_{1,m}$}
...
{$Symb_{1,n}$ $Symb_{2,n}$ ... $Symb_{n,m}$}}

Function Transposes the list-matrix.

Notes The matrix may have any dimension.

See Also SQUARE?

Applications MATMENU, PRSYMB

Examples

1: {{ 1 2 }
{ X -1 }
{ '-X' 5 }}

TRNSP **[ENTER]**

1: {{ 1 X '-X' }
{ 2 -1 5 }}

TRUE

Category	Type Conversion
Affected by flag	none
Input	
Output	1: External (TRUE)
Function	Pushes on the stack the system boolean TRUE.
Notes	System booleans are machine language routine addresses which merely return themselves when evaluated. The command →TorF turns a system boolean into a real boolean.
See Also	EXT→, →EXT, TRUE, →TorF

VER$

Category	Rom Version
Affected by flag	none
Input	
Output	1: "SMRT 1:B"
Function	Returns the current version of the SmartROM.
Notes	It can be useful for creating programs running on different versions of the ROM.

→Xlib

Category Type Conversion

Affected by flag none

Input

2: LID
1: Num

Output

1: XLIB LID Num

Function

Pushes on the stack the External Library name specified.

Notes

When you put on the stack a XLIB object having a corresponding text name, the stack display shows it directly. If you push a so-called hidden command, there is no way to get a text name for that object. There are two ways to know if a XLIB is referenced:

By evaluating it: Very dangerous!

By calling the entry point # 07E99 with SYSEVAL
The last method is the safest.

If an object is referenced by the XLIB name specified, entry point #07E99 returns it along with the system boolean TRUE. Otherwise it returns FALSE.

If you try to evaluate an undefined XLIB name , you will get an error message. On the other hand if it refers to a routine and the stack does not contain proper arguments, you have a big chance to loose memory data.

See Also →EXT, EXT→

Examples

2: 2
1: 81

→Xlib **[ENTER]**

1: SIN

2: 821
1: 45

→Xlib **[ENTER]**

1: MGET

2: 821
1: 246

→Xlib **[ENTER]**

1: XLIB 821 246 Don't worry !

XLVLS

Category Stack Manipulation

Affected by flag none

Input

...:	...
P+2:	obj_p
...:	...
U+2:	obj_u
...:	...
3:	obj_1
2:	p
1:	u

Output

...:	...
P :	obj_u
...:	...
U :	obj_p
...:	...
1 :	obj_1

Function Exchanges levels p and u.

See Also RDROP, RDUP, SHIFT

Examples

5:	"ABCDE"
4:	1
3:	2
2:	3
1:	"hello"

4 5 XLVLS **[ENTER]**

5:	1
4:	"ABCDE"
3:	2
2:	3
1:	"hello"

Appendix A

Care of the SmartROM

The SmartROM does not require maintenance.

Limited One Year Warranty

The SmartROM is warranted by Smart Technology against defects in materials and workmanship for one year from the date of original purchase. Warranty is automatically transferred to new owner if you sell the product or give it as a gift and remains in effect for the original one-year period. During the warranty period, we will repair or, at our option, replace at no charge a product that proves to be defective, provided you return the product, shipping prepaid, to Smart Technology.

The warranty does not apply if the product has been damaged by accident or misuse or as the result of service or modification by other than Smart Technology.

No other express warranty is given.

Smart Technology makes no express or implied warranty with regard to the software furnished. Programs are made available solely on an 'as is' basis and the entire risk as to its quality and performance is with the user. Should documentation and programs prove to be defective, the user (and not Smart Technology or any other party) shall bear the entire cost of all necessary correction and all incidental or consequential damages. Smart Technology shall not be liable for any incidental or consequential damages in connection with or arising out of the furnishing, use or performance of the documentation and programs.

Service Center

Whether your unit is under warranty or not you can ship it for repair to our Service Center. If your warranty has expired, there will be a charge for the repair and for shipping costs.

The Service Center is located in Modena ITALY.

SMART TECHNOLOGY
Via Varese 67
41100 Modena, ITALY
phone 059-440404
fax 059-304490

Normally Your unit will be repaired within five (5) working days of receipt.

Service Repair Charge

There is a standard repair price (STREP) for out-of-warranty repairs.

Out-of-Warranty units returned after repair are warranted for a limited 90 days period against defects in materials or workmanship.

Shipping Instructions

If your unit requires service, please follow these shipping instructions:

- Include a description of the problem detected.
- If under warranty, include documentation proving the date of purchase or repair.
- Ship the unit in a protective packaging to prevent additional damages.

Shipping to Smart Technology is at your charge. Shipping costs to return the unit are paid by Smart Technology and will be included in the bill. On out-of-warranty repairs, the unit will be returned C.O.D.

Technical Assistance

Smart Technology is committed to provide strong after-sale customer support. If you need specific information on this product or technical help on HP Calculators, you can call the number given above.

Appendix B

Objects structure

Object classification proposed herein follows the order estabilished by the system function TYPE. For each type of object internal code used by dispatching routines is given too.

Each object is composed of a Prolog, i.e. the header of the object that determines its behavior during direct or indirect evaluation and its data body along with its structure, total dimensions and characteristics.

Dimensions (length in nibbles) are given in the form: ($Prolog_5$) ... ($Data_n$). Each different item represents a logic unit whose length is specified in nibbles by the subscript.

Note that the 48 arranges data in memory in reverse order, so that the prolog is written with the least significant nibble coming first. All examples are given as they would appear in memory during a Memory Scanner session. If you don't know what the Memory Scanner is, ***save important data first*** and press **[ON] [D] [BKSP]**.
[+] and **[-]** let you shift back and forth by 1 nibble.
[*] and [/] let you skip over 256 nibbles.
[↑] and [↓] let you skip over 4K nibble at a time.
Never use **[EVAL]**! For more information on Memory Scanner, check BBS contents on this topic or read the HP-48 Handbooks as those of James Donnelly and Bill Wickes.

Real number

Type 0

Internal Type <1h>

Prolog <02933h>

Structure ($Prolog_5$) ($Exponent_3$) ($Mantissa_{12}$) ($Sign_1$)

Dimensions 21

Data (Exponent3)

BCD Exponent in ten's complement (-500 to 500)

($Mantissa_{12}$)

BCD Mantissa

($Sign_1$)

Sign: 0 = positive, 9 = negative

Example

0	is equal to 339200000000000000000
pi	is equal to 339200009535629514130
-1	is equal to 332900000000000000019
-11	is equal to 332901000000000000119
-.5	is equal to 332909990000000000059

Complex Number

Type	1
Internal Type	<2h>
Prolog	<02977h>
Structure	($Prolog_5$) ($Exponent_3$) ($Mantissa_{12}$) ($Sign_1$) ($Exponent_3$) ($Mantissa_{12}$) ($Sign_1$)
Dimensions	37
Data	The Real part is composed by the first number while the imaginary part comes next. Number representation is the same as for reals.

String

Type	2
Internal Type	<3h>
Prolog	<02A2Ch>
Structure	($Prolog_5$) ($offset_5$) characters.
Dimensions	5 + offset
Data	The total number of characters is (offset-5)/2. (offset-5) must always be an even number.
Notes	Characters are stored byte reversed.
Example	"CIAO" is equal to: C2A20D000034E414F4 "" is equal to : C2A2050000

Real array

Type	3
Internal Type	<4h>
Prolog	<02E48h>
Structure	($Prolog_5$) ($offset_5$) (Real $Prolog_5$) (n-dim_5) (dim_1) ... (dim_n) ($R_{1...1}$) ... (R_{dim1} dim_2...dim_n)
Dimensions	5 + offset.
Data	Matrices are stored in row major order incrementing the rightmost counter faster.
Notes	It is possible to create n-dimensional order arrays. However there are no provisions in the system for handling individual elements when n is greater than 2. If you put on the stack a 3-dimensional array, you will get only "Array of reals".

Complex array

Type	4
Internal Type	<4h>
Prolog	<02E48h>
Structure	($Prolog_5$) ($offset_5$) (Complex Prolog) (n-dim) (dim_1) ... (dim_n) ($Cdim_1$ 1 ... 1) ... ($Cdim_1$ dim_2 ... dim_n)
Dimensions	5 + offset.
Data	Matrices are stored in row major order incrementing the rightmost counter faster.
Notes	It is possible to create n-dimensional order matrices. However there are no provisions in the system for handling individual elements when n is greater than 2. If you put on the stack a 3-dimensional array, you will get only "Array of complex".

Array

Type 4

Internal Type <4h>

Prolog <02E48h>

Structure ($Prolog_5$) ($offset_5$) (Data Prolog) (n-dim) (dim_1) ... (dim_n) ($Data_1$) ... ($Data_n$)

Dimensions 5 + offset.

Notes You can create non-numeric arrays for storing type-homogeneous data. Unfortunately there are no provisions to handle efficiently this kind of objects. Error Messages in HIDE area are stored in several one-dimensional string arrays. If you want to store some data preserving it from editing, non-numeric arrays are a good place because of their inaccessibility.

List

Type 5

Internal Type <5h>

Prolog <02A74h>

Structure ($Prolog_5$) Obj_1 Obj_2 ... Obj_n (End_5)

Dimensions $5 + Length(Obj_1) + Length(Obj_2) + ... + Length(Obj_n) + 5$

Data A list is a composite object whose body is a sequence of objects or pointers to objects terminated by a special pointer <0312Bh>.

Notes The list object is similar to program objects and symbolic expressions. If you want to translate a Symbolic expression into a List, change its prolog to <02A74h>. When you evaluate a List or a Symbolic Expression through EVAL, special code is called to change the prolog of the composite object into the prolog of a program object.

Example

{}	is equal to	47A20B2130
{ 1 "" }	is equal to	47A209C2A2FD550B2130
	or to	47A203392000000000000000010C2A2050000B2130

Global name

Type	6
Internal Type	<6h>, <Ah>
Prolog	<02E48h>
Structure	($Prolog_5$) ($Length_2$) characters.
Dimensions	7 + (Length) * 2
Notes	Maximum lenght of an indentifier is 255 characters with no restrictions on the name. However there are restrictions due to the parser safety rules, which prevents you from creating names conflicting with reserved variables used by the system.
Example	'MARK is equal to 84E205072D41425B4

Local name

Type	7
Internal Type	<7h>
Prolog	<02E6Dh>
Structure	$(Prolog_5)$ $(Length_2)$ characters.
Dimensions	7 + (Length) * 2
Notes	See on the previous page.
Example	matA is equal to D6E2040D6164714

Program

Type 8

Internal Type <8h>

Prolog <02D9Dh>

Structure ($Prolog_5$) Obj_1 Obj_2 ... Obj_n (End_5)

Dimensions 5 + Length(Obj_1) + Length(Obj_2) + ... + Length(Obj_n) + 5

Notes A program is a composite object whose body is a sequence of objects or pointers to objects terminated by a special pointer 8Bh. Main difference with List and Symbolic lies in its direct execution capability.

Example

Internal program performing Rot Dup2 without stack checking:

D9D2059230CA130B2130

Algebraic

Type	9
Internal Type	<9h>, <Ah>
Prolog	<02AB8h>
Structure	$(Prolog_5)\ Obj_1\ Obj_2 \ldots Obj_n\ (End_5)$
Dimensions	$5 + Length(Obj_1) + Length(Obj_2) + \ldots + Length(Obj_n) + 5$
Notes	Symbolic objects are similar to program objects and symbolic expressions. If you want to translate a Symbolic expression into a List, change its prolog to A74h. When you evaluate a List or a Symbolic Expression through EVAL, special code is called to change the prolog of the composite object into the prolog of a program object.

Binary integer

Type 10

Internal Type <Bh>

Prolog <02A4Eh>

Structure $(Prolog_5)$ $(offset_5)$ nibbles

Dimensions 5 + offset

Data Raw nibbles.

Notes Binary integers may exceed 64 bit width. However internal arithmetic routines are tailored for handling 64 bit max integers. Other internal routines (like Append or Size) work well independently of integer length. A curious aspect of a Library structure is that the execution, decompile and text tables are stuffed in huge binary integers objects. User defined binary integers generally have a standard length of 16 nibbles independently of the actual wordsize.

Example

This is the shortest form of

#1234 wich translates to : E4A20900004321

Graphic object

Type 11

Internal Type <Ch>

Prolog <02B1Eh>

Structure ($Prolog_5$) ($Offset_5$) ($Rows_5$) ($Columns_5$) nibbles

Dimensions 5 + offset.

Data Graphics objects are stored in row major order using a bit for each pixel on a byte-aligned scheme. Thus, each row of pixel must have an even number of nibbles eventually padding with garbage bits the last byte. The least significant bit of each nibble represents the leftmost pixel of 4 pixel block.

Example

Take the character A in the font ROM8x14:

E1B20B2000E000080000000000183C66C6CEF6C6C6C0
00000

```
1       ........        00
2       ........        00
3       ....*...        01
4       ...***..        83
5       ..**.**.        C6
6       .**...**        6C
7       .**...**        6C
8       .*******        EF
9       .**...**        6C
A       .**...**        6C
B       .**...**        6C
C       ........        00
D       ........        00
E       ........        00
```

Tagged object

Type	12
Internal Type	<Dh>
Prolog	<02AFC>
Structure	($Prolog_5$) ($Length_2$) characters Obj
Dimensions	7 + (Length) * 2 + Length(Obj)
Notes	Tagged objects does not inherit the behavior of the ancestor type unless you make a recursive call to the routine being executed after deleting the tag.
Example	PIGREEK: 3.14159265359 is represented by : CFA2070059474255454B4339200009535629514130

Unit

Type 13

Internal Type <Eh>

Prolog <02ADAh>

Structure ($Prolog_5$) (value) (string) ... (string) (oper5) ... (oper5) (End_5)

Dimensions 5 + 21 + Length(string)+ ...+ Length(string) + 5 * num(oper) + 5

Example 1.5129_m^2 is equal to:

ADA203392000000000000921510C2A20700006ED2A227
B0168B01B2130

Xlib name

Type	14
Internal Type	<0Fh>
Prolog	<02E92h>
Structure	($Prolog_5$) (LID_3) (Num_3)
Dimensions	11
Data	External Library Names are uniquely identified by a Library Identification Number and a library-local command number. System User commands are double-face. They have a fixed address pointer which allows faster execution and memory-saving storage. However if you put a command on the stack and store it in a global name in order to send it to a PC, special code in the STO command changes the address pointer of the command into the respective External Library Name to avoid transferring rom-based code.

Directory

Type 15

Internal Type <2Fh>

Prolog <02A96h>

Structure (Prolog$_5$) (Attach 3) (offset1) (00000) Namen Obj$_n$ (offset n+1) Namen-1 Obj$_n$-1 (offsetn) ... Name1 Obj$_1$ (offset2)

Dimensions 8 + offset1 + 5

Data Object are stored in reverse order. (offset$_1$) points to the field (offset$_2$) at the bottom of the directory where lies the first object. A sequence of backward offsets lets you jump like a frog till the last object at the top of the directory. The last offset field (00000) marks the end of the chain. The Attach field retains the Library identification number of one Library. In the Home directory this field counts the number of library actually attached to the HOME dir.

Library

Type 16

Internal Type <8Fh>

Prolog <02B40h>

Structure ($Prolog_5$) (offset5) (Length 2) Characters (LID_3) (Offset TexTTbl 5) (Offset MsgTbl 5) (Offset LinkTbl 5) (Offset Conf 5) Nibbles (Cksum 4)

Dimensions 5 + offset

Data

(Length 2) Characters	Library Title.
(LID_3)	Library IDentification number.
(Offset $TextTbl_5$)	Offset to a binary integer containing command names text. The table begins with 16 field of 5 nibbles (80 nibbles total) each one of them pointing to the first command of a given length. The first field points to the first command (in alphabetical order) of length 1.
(Offset $MsgTbl_5$)	Offset to the table of messages. The table is contained in a string-array.
(Offset $LinkTbl_5$)	Offset to a binary integer containing all the executable code of commands.
(Offset $Config_5$)	Offset to the configuration program of the library.
Nibbles	Library contents.
(Cksum 4)	Checksum.

Backup

Type	17
Internal Type	<9Fh>
Prolog	<02B62h>
Structure	($Prolog_5$) (offset 5) (Length 2) characters nibbles
Dimensions	5 + offset
Notes	Backup objects normally exist only in memory ports.

System function

Type 18

Internal Type <8h>

Prolog <02E92h>

Structure ($Prolog_5$) (LID_3) (Num_3)

Dimensions 11

Data A rom-based program is recognized as function when a special code precedes its execution address. Formerly it is a normal program object but it receives special handling by the parser and decompile routines when you enter formulas in algebraic style. The process of recognizing algebraic functions is a bit tricky and this is not the best place where place an exhaustive explanation. There are several bits specifying analytic properties like differentiability and others specifing if the program is a command or a function and where to place the decompiled text (before, between or after). The study of USAG program can be very helpful if you are interested.

Example

SIN (system function) is identified by XLIB 2 81. Its special header is CC0 and precedes the LID program in ROM. If you scan memory a little before location B4ACh, you will see the following sequence :

CC0200150D9D20....

CC0 means integrable, invertible, differentiable, function.

Thanks to this encoding system, adding external function is very, very hard.

System Command

Type	19
Internal Type	<8h>
Prolog	<02E92h>
Structure	($Prolog_5$) (LID_3) (Num_3)
Dimensions	11
Data	A rom-based program is recognized as built-in command when a special code precedes its execution address. Most commands have a single nibble header (added to the six specifying the Library ID and number) whose value is 8. It seems that the most significant bit of this nibble play a key-role in the game. If this bit is 0, the program is some kind of function. The study of USAG program can be very helpful if you are interested.
Example	STO (system command) is identified by XLIB 2 341.

System binary

Type	20
Internal Type	<1Fh>
Prolog	<02911h>
Structure	($Prolog_5$) ($Nibble_5$)
Dimensions	10
Data	System binaries are the most used numeric entities throughout the Operating System. Providing a faster throughput than real numbers and smaller storage requirements, they are the optimum choice for the system programmer.
Example	FFFFFh is equal to 11920FFFFF 12345h is equal to 1192054321

Long real

Type	21
Internal Type	<3Fh>
Prolog	<02955h>
Structure	($Prolog_5$) ($Exponent_5$) ($Mantissa_{15}$) ($Sign_1$)
Dimensions	26
Data	($Exponent_5$)
	Exponent BCD in ten's complement (from -50000 to 50000)
	(Mantissa 15)
	BCD Mantissa
	($Sign_1$)
	Sign: 0 = positive, 9 = negative
Example	Extended precision PI is equal to
	55920000009798535629514130

Long complex

Type	22
Internal Type	<4Fh>
Prolog	<0299Dh>
Structure	($Prolog_5$) ($Exponent_5$) ($Mantissa_{15}$) ($Sign_1$) ($Exponent_5$) ($Mantissa_{15}$) ($Sign_1$)
Dimensions	47
Data	Real part comes first, next the imaginary part. The number encoding system is the same as for Extended Reals.

Linked array

Type 23

Internal Type <5Fh>

Prolog <02A0Ah>

Structure ($Prolog_5$) ($Offset_5$) (Data $Prolog_5$) (n-dim) (dim_1) ... (dim_n) ($pointer_{1,...,1}$) ... ($pointer_{dim\ 1,...,\ dim\ n}$)

Dimensions 5 + Offset.

Data This is one of the most esoteric data structures built in the 48. In the 256K of the operating system, there is no evidence of its existence. It seems that this structure is suitable for sparse arrays because a missing element is represented by a 00000 pointer. If you have more amazing news about it, we will be glad to hear from you.

Character

Type	24
Internal Type	<6Fh>
Prolog	<029BFh>
Structure	($Prolog_5$) ($Byte_2$)
Dimensions	7
Data	Contains a single-byte (a character) byte reversed.
Example	Letter A is represented by : FB92014

Code

Type 25

Internal Type <7Fh>

Prolog <02DCCh>

Structure ($Prolog_5$) ($Offset_5$) Nibbles

Dimensions 5 + Offset

Data The code starts at the first nibble of the body. Execution is transferred to this location by the prolog of the object.

Notes See the Appendix C for an explanation of the Saturn Assembly Language.

Example The routine DONOTHING:

```
A=DAT0  A
D0=D0+  5
PC=(A)
```

Read the address of next object Updates Thread pointer. Skip to next execution address.

written in memory:

CCD20F0000142164808C

Library data

Type	26
Internal Type	<AFh>
Prolog	<02B88h>
Structure	($Prolog_5$) ($offset_5$) (LID_3) (num_3) Obj_1 Obj_2 ... Obj_n (End_5)
Dimensions	5 + Offset
Data	Libraries may use this kind of object when they want to preserve data from editing till the next library call. Data appear on the stack merely as 'Library Data'.

Address

Type 27

Internal Type 0h

Prolog <hhhhh>

Structure ($Prolog_5$)

Dimensions 5

Data Any object not mentioned before falls in this category. Whether it is possible (and safe) to add new object types to the HP-48 RPL is not clear. Theoretically it should be. Under normal circumstancies, this category represents atomic threads. An atomic thread is an address where machine language code begins. Built in machine language routines (callable as RPL routines) have the following form:

```
Addr    Addr+5

Prolog  Nibbles....
```

Prolog is a 5 nibble offset to the location where Saturn codes begin. By convention it should be Addr+5, but some routines violate this convention to implement hyper-compact Dup'n Go routines.

Appendix C

The Saturn microprocessor

The microprocessor being used in the HP-48 family of calculators is the evolution of the HP-71 handheld computer and HP-28 pocket calculator. It is a low power consumption CPU, optimized for BCD calculation. Memory Addressing limit is, theoretically, 1 M nibbles, that is 512 Kbytes. With the help of bank switching software techniques the 1Mb nibble barrier can be broken . Tripod Data Systems memory cards reach 512K bytes on a single card, by exploiting these software tricks.

The Saturn Assembly Language Instruction Set explained on the following pages follows HP's original Saturn Assembler Internal Design Specification as it is documented in the HP-71 Hardware Internal Design Specification. Mnemonics added after the release of the HP-71 documentation are the same accepted by HP's unsupported Saturn Compiler available at EduCALC (Goodies disc n°4).

Microprocessor's registers

The CPU is organized in several registers with different characteristics and power.

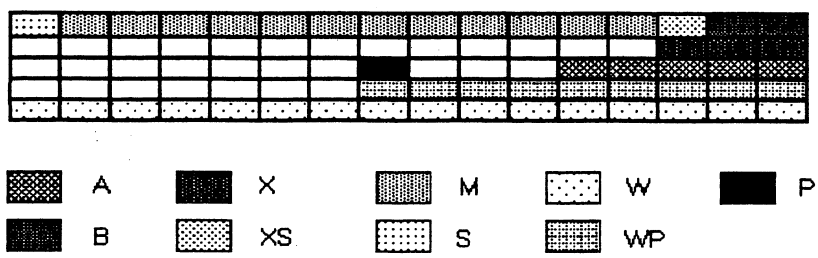
Scratch Registers

There are five temporary registers (Scratch Registers) of 64 bit length called R0, R1, R2, R3 and R4. These registers serve as storage area during operations. It is possible to read and write on specific fields ranging from 1 nibble up to the whole register.

Arthmetic Registers

Four arithmetic registers (A, B, C and D) of 64 bit lentgh. These are the working registers. Each one of these registers can be accessed through the following fields:

Field Name	Length in nibbles	Length in bits	Meaning
W	16	64	WORD
A	5	20	ADDRESS
B	2	8	BYTE
M	12	48	MANTISSA
S	1	4	SIGN
X	3	12	EXPONENT
XS	1	4	EXPONENT SIGN
P	1	4	POINTER
WP	P+1	(P+1)*4	WORD THROUGH Pointer



Register C is the most powerful of the four and allows the most flexible exchange of data with others. Register D is the least powerful because it can communicate only with C.

Control registers

ST — Status Register. It si a 16 bit register, where the four most significant bits (from bit 12 to 15) are normally accessed only by the operating system.

PC — Program Counter, 20 bit register controlling the execution flow. It is accessible indirectly.

RSTK — Return Stack. It is a (LIFO) stack with 8 levels of 20 bit where subroutine return addresses are automatically pushed. It can be accessed also through register C.

P — Pointer, 4 bit register specifying a nibble within arithmetic or scratch registers. Register P determines also the length of field WP.

D0 — Address register. 20 bit register for addressing data in memory.

D1 — Address register. 20 bit register for addressing data in memory.

IN — Input Register. 16 bit read-only register used by the system to control the keyboard.

OUT — Ouptut Register. 12 bit write-only register used by the system to enable the keyboard and the beeper.

SB — Sticky Bit. 1 bit field of Hardware Status Register whose value is determined by right shift operations of arithmetic registers.

SR — Service Request Bit. 1 bit field of Hardware Status Register whose value is determined by external events.

MP — Module Pulled Bit. 1 bit field of Hardware Status Register whose value is determined by the pulling or pushing of hardware modules.

XM — External Module Missing Bit. 1 bit field of Hardware Status Register whose value is determined by software.

Saturn Assembly Language Mnemonics

Legenda

The conventions explained herein are used in the following pages to represent parameters and values which may vary in a certain range.

Symbol	Meaning
a	Hex digit which represents a number in the range 0-7 used in the instruction opcode.
b	Hex digit which represents a number in the range 8-F used in the instruction opcode.
d	Decimal digit whose value ranges from 1 to 16. The assembler diminishes it automatically by one.
fs	Field selection (B, X, XS and so on).
hh...h	Up to 16 hex digits.

Field Selection Table

Often in the instruction table fields are referred to as fs. In order to isolate a specific field, you need to know exactly the

value of a specific digit in the opcode. The following table summarizes the convention.

Field	Opcode value		Length
	(a)	(b)	(d)
P	0	8	1
WP	1	9	(P)+1
XS	2	A	1
X	3	B	3
S	4	C	1
M	5	D	12
B	6	E	2
W	7	F	16

Opcodes vs Mnemonics

This is the complete reference to all Saturn opcodes.

Hex	Mnemonic	Field
00	RTNSXM	
01	RTN	
02	RTNSC	
03	RTNCC	
04	SETHEX	
05	SETDEC	
06	RSTK=C	
07	C=RSTK	
08	CLRST	
09	C=ST	
0A	ST=C	
0B	CSTEX	
0C	P=P+1	
0D	P=P-1	
0EF0	A=A&B	A
0EF1	B=B&C	A
0EF2	C=C&A	A
0EF3	D=D&C	A
0EF4	B=B&A	A
0EF5	C=C&B	A
0EF6	A=A&C	A
0EF7	C=C&D	A
0EF8	A=A!B	A
0EF9	B=B!C	A
0EFA	C=C!A	A
0EFB	D=D!C	A
0EFC	B=B!A	A
0EFD	C=C!B	A
0EFE	A=A!C	A
0EFF	C=C!D	A

0Ea0	A=A&B	fs
0Ea1	B=B&C	fs
0Ea2	C=C&A	fs
0Ea3	D=D&C	fs
0Ea4	B=B&A	fs
0Ea5	C=C&B	fs
0Ea6	A=A&C	fs
0Ea7	C=C&D	fs
0Ea8	A=A!B	fs
0Ea9	B=B!C	fs
0EaA	C=C!A	fs
0EaB	D=D!C	fs
0EaC	B=B!A	fs
0EaD	C=C!B	fs
0EaE	A=A!C	fs
0EaF	C=C!D	fs
0F	RTI	
100	R0=A	
101	R1=A	
102	R2=A	
103	R3=A	
104	R4=A	
108	R0=C	
109	R1=C	
10A	R2=C	
10B	R3=C	
10C	R4=C	
110	A=R0	
111	A=R1	
112	A=R2	
113	A=R3	
114	A=R4	
118	C=R0	
119	C=R1	
11A	C=R2	
11B	C=R3	
11C	C=R4	
120	AR0EX	
121	AR1EX	
122	AR2EX	
123	AR3EX	
124	AR4EX	
128	CR0EX	
129	CR1EX	
12A	CR2EX	
12B	CR3EX	
12C	CR4EX	
130	D0=A	
131	D1=A	
132	AD0EX	
133	AD1EX	
134	D0=C	
135	D1=C	
136	CD0EX	
137	CD1EX	
138	D0=AS	
139	D1=AS	

13A	AD0XS	
13B	AD1XS	
13C	D0=CS	
13D	D1=CS	
13E	CD0XS	
13F	CD1XS	
140	DAT0=A	A
141	DAT1=A	A
142	A=DAT0	A
143	A=DAT1	A
144	DAT0=C	A
145	DAT1=C	A
146	C=DAT0	A
147	C=DAT1	A
148	DAT0=A	B
149	DAT1=A	B
14A	A=DAT0	B
14B	A=DAT1	B
14C	DAT0=C	B
14D	DAT1=C	B
14E	C=DAT0	B
14F	C=DAT1	B
150a	DAT0=A	fs
151a	DAT1=A	fs
152a	A=DAT0	fs
153a	A=DAT1	fs
154a	DAT0=C	fs
155a	DAT1=C	fs
156a	C=DAT0	fs
157a	C=DAT1	fs
158x	DAT0=A	d
159x	DAT1=A	d
15Ax	A=DAT0	d
15Bx	A=DAT1	d
15Cx	DAT0=C	d
15Dx	DAT1=C	d
15Ex	C=DAT0	d
15Fx	C=DAT1	d
16x	D0=D0+	n
17x	D1=D1+	n
18x	D0=D0-	n
19nn	D0=(2)	hhhhh
1Annnn	D0=(4)	hhhhh
1Bnnnnn	D0=(5)	hhhhh
1Cx	D1=D1-	n
1Dnn	D1=(2)	hhhhh
1Ennnn	D1=(4)	hhhhh
1Fnnnnn	D1=(5)	hhhhh
2n	P=	n
3nh...h	LCHEX hh...h	
400	RTNC	
420	NOP3	
4oo	GOC	
500	RTNNC	
5oo	GONC	
6300	NOP4	
64000	NOP5	

6ooo	GOTO	
7ooo	GOSUB	
800	OUT=CS	
801	OUT=C	
802	A=IN	
803	C=IN	
804	UNCFNG	
805	CONFIG	
806	C=ID	
807	SHUTDN	
8080	INTON	
80810	RSI	
8082nhh...h	LAHEX	hh...h
8083	BUSCB	
8084x	ABIT=0	x
8085x	ABIT=1	x
8086xoo	?ABIT0	x
8087xoo	?ABIT1	x
8088x	CBIT=0	x
8089x	CBIT=1	x
808Axoo	?CBIT=0	x
808Bxoo	?CBIT=1	x
808C	PC=(A)	
808D	BUSCD	
808E	PC=(C)	
808F	INTOFF	
809	C+P+1	
80A	RESET	
80B	BUSCC	
80Cn	C=P	n
80Dn	P=C	n
80E	SREQ?	
80Fn	CPEX	n
810	ASLC	
811	BSLC	
812	CSLC	
813	DSLC	
814	ASRC	
815	BSRC	
816	CSRC	
817	DSRC	
818a0d	A=A+d	fs
818a1d	B=B+d	fs
818a2d	C=C+d	fs
818a3d	D=D+d	fs
818F0d	A=A+d	A
818F1d	B=B+d	A
818F2d	C=C+d	A
818F3d	D=D+d	A
818a8d	A=A-d	fs
818a9d	B=B-d	fs
818aAd	C=C-d	fs
818aBd	D=D-d	fs
818F8d	A=A-d	A
818F9d	B=B-d	A
818FAd	C=C-d	A
818FBd	D=D-d	A

```
819a0       ASRB        fs
819a1       BSRB        fs
819a2       CSRB        fs
819a3       DSRB        fs
819F0       ASRB        A
819F1       BSRB        A
819F2       CSRB        A
819F3       DSRB        A
81Aa00      R0=A        fs
81Aa01      R1=A        fs
81Aa02      R2=A        fs
81Aa03      R3=A        fs
81Aa04      R4=A        fs
81Aa08      R0=C        fs
81Aa09      R1=C        fs
81Aa0A      R2=C        fs
81Aa0B      R3=C        fs
81Aa0C      R4=C        fs
81AF00      R0=A        A
81AF01      R1=A        A
81AF02      R2=A        A
81AF03      R3=A        A
81AF04      R4=A        A
81AF08      R0=C        A
81AF09      R1=C        A
81AF0A      R2=C        A
81AF0B      R3=C        A
81AF0C      R4=C        A
81Aa10      A=R0        fs
81Aa11      A=R1        fs
81Aa12      A=R2        fs
81Aa13      A=R3        fs
81Aa14      A=R4        fs
81Aa18      C=R0        fs
81Aa19      C=R1        fs
81Aa1A      C=R2        fs
81Aa1B      C=R3        fs
81Aa1C      C=R4        fs
81AF10      A=R0        A
81AF11      A=R1        A
81AF12      A=R2        A
81AF13      A=R3        A
81AF14      A=R4        A
81AF18      C=R0        A
81AF19      C=R1        A
81AF1A      C=R2        A
81AF1B      C=R3        A
81AF1C      C=R4        A
81Aa20      AR0EX       fs
81Aa21      AR1EX       fs
81Aa22      AR2EX       fs
81Aa23      AR3EX       fs
81Aa24      AR4EX       fs
81Aa28      CR0EX       fs
81Aa29      CR1EX       fs
81Aa2A      CR2EX       fs
81Aa2B      CR3EX       fs
```

```
81Aa2C      CR4EX       fs
81AF20      AR0EX       A
81AF21      AR1EX       A
81AF22      AR2EX       A
81AF23      AR3EX       A
81AF24      AR4EX       A
81AF28      CR0EX       A
81AF29      CR1EX       A
81AF2A      CR2EX       A
81AF2B      CR3EX       A
81AF2C      CR4EX       A
81B2        PC=A
81B3        PC=C
81B4        A=PC
81B5        C=PC
81B6        APCEX
81B7        CPCEX
81C         ASRB
81D         BSRB
81E         CSRB
81F         DSRB
821         XM=0
822         SB=0
824         SR=0
828         MP=0
82F         CLRHST
831yy       ?XM=0
832yy       ?SB=0
834yy       ?SR=0
838yy       ?MP=0
84n         ST=0   n
85n         ST=1   n
86nyy       ?ST=0       n
87nyy       ?ST=1       n
88nyy       ?P#         n
89nyy       ?P=         n
8A0yy       ?A=B        A
8A1yy       ?B=C        A
8A2yy       ?C=A        A
8A3yy       ?D=C        A
8A4yy       ?A#B        A
8A5yy       ?B#C        A
8A6yy       ?C#A        A
8A7yy       ?D#C        A
8A8yy       ?A=0        A
8A9yy       ?B=0        A
8AAyy       ?C=0        A
8AByy       ?D=0        A
8ACyy       ?A#0        A
8ADyy       ?B#0        A
8AEyy       ?C#0        A
8AFyy       ?D#0        A
8B0yy       ?A>B        A
8B1yy       ?B>C        A
8B2yy       ?C>A        A
8B3yy       ?D>C        A
8B4yy       ?A<B        A
```

8B5yy	?B<C	A
8B6yy	?C<A	A
8B7yy	?D<C	A
8B8yy	?A>=B	A
8B9yy	?B>=C	A
8BAyy	?C>=A	A
8BByy	?D>=C	A
8BCyy	?A<=B	A
8BDyy	?B<=C	A
8BEyy	?C<=A	A
8BFyy	?D<=C	A
8Coooo	GOLONG	
8Daaaaa	GOVLNG	
8Eoooo	GOSUBL	
8Faaaaa	GOSBVL	
9a0yy	?A=B	fs
9a1yy	?B=C	fs
9a2yy	?C=A	fs
9a3yy	?D=C	fs
9a4yy	?A#B	fs
9a5yy	?B#C	fs
9a6yy	?C#A	fs
9a7yy	?D#C	fs
9a8yy	?A=0	fs
9a9yy	?B=0	fs
9aAyy	?C=0	fs
9aByy	?D=0	fs
9aCyy	?A#0	fs
9aDyy	?B#0	fs
9aEyy	?C#0	fs
9aFyy	?D#0	fs
9b0yy	?A>B	fs
9b1yy	?B>C	fs
9b2yy	?C>A	fs
9b3yy	?D>C	fs
9b4yy	?A<B	fs
9b5yy	?B<C	fs
9b6yy	?C<A	fs
9b7yy	?D<C	fs
9b8yy	?A>=B	fs
9b9yy	?B>=C	fs
9bAyy	?C>=A	fs
9bByy	?D>=C	fs
9bCyy	?A<=B	fs
9bDyy	?B<=C	fs
9bEyy	?C<=A	fs
9bFyy	?D<=C	fs
Aa0	A=A+B	fs
Aa1	B=B+C	fs
Aa2	C=C+A	fs
Aa3	D=D+C	fs
Aa4	A=A+A	fs
Aa5	B=B+B	fs
Aa6	C=C+C	fs
Aa7	D=D+D	fs
Aa8	B=B+A	fs
Aa9	C=C+B	fs

AaA	A=A+C	fs
AaB	C=C+D	fs
AaC	A=A-1	fs
AaD	B=B-1	fs
AaE	C=C-1	fs
AaF	D=D-1	fs
Ab0	A=0	fs
Ab1	B=0	fs
Ab2	C=0	fs
Ab3	D=0	fs
Ab4	A=B	fs
Ab5	B=C	fs
Ab6	C=A	fs
Ab7	D=C	fs
Ab8	B=A	fs
Ab9	C=B	fs
AbA	A=C	fs
AbB	C=D	fs
AbC	ABEX	fs
AbD	BCEX	fs
AbE	CAEX	fs
AbF	DCEX	fs
Ba0	A=A-B	fs
Ba1	B=B-C	fs
Ba2	C=C-A	fs
Ba3	D=D-C	fs
Ba4	A=A+1	fs
Ba5	B=B+1	fs
Ba6	C=C+1	fs
Ba7	D=D+1	fs
Ba8	B=B-A	fs
Ba9	C=C-B	fs
BaA	A=A-C	fs
BaB	C=C-D	fs
BaC	A=B-A	fs
BaD	B=C-B	fs
BaE	C=A-C	fs
BaF	D=C-D	fs
Bb0	ASL	fs
Bb1	BSL	fs
Bb2	CSL	fs
Bb3	DSL	fs
Bb4	ASR	fs
Bb5	BSR	fs
Bb6	CSR	fs
Bb7	DSR	fs
Bb8	A=-A	fs
Bb9	B=-B	fs
BbA	C=-C	fs
BbB	D=-D	fs
BbC	A=-A-1	fs
BbD	B=-B-1	fs
BbE	C=-C-1	fs
BbF	D=-D-1	fs
C0	A=A+B	A
C1	B=B+C	A
C2	C=C+A	A

C3	D=D+C	A
C4	A=A+A	A
C5	B=B+B	A
C6	C=C+C	A
C7	D=D+D	A
C8	B=B+A	A
C9	C=C+B	A
CA	A=A+C	A
CB	C=C+D	A
CC	A=A-1	A
CD	B=B-1	A
CE	C=C-1	A
CF	D=D-1	A
D0	A=0	A
D1	B=0	A
D2	C=0	A
D3	D=0	A
D4	A=B	A
D5	B=C	A
D6	C=A	A
D7	D=C	A
D8	B=A	A
D9	C=B	A
DA	A=C	A
DB	C=D	A
DC	ABEX	A
DD	BCEX	A
DE	CAEX	A
DF	DCEX	A
E0	A=A-B	A
E1	B=B-C	A
E2	C=C-A	A
E3	D=D-C	A
E4	A=A+1	A
E5	B=B+1	A
E6	C=C+1	A
E7	D=D+1	A
E8	B=B-A	A
E9	C=C-B	A
EA	A=A-C	A
EB	C=C-D	A
EC	A=B-A	A
ED	B=C-B	A
EE	C=A-C	A
EF	D=C-D	A
F0	ASL	A
F1	BSL	A
F2	CSL	A
F3	DSL	A
F4	ASR	A
F5	BSR	A
F6	CSR	A
F7	DSR	A
F8	A=-A	A
F9	B=-B	A
FA	C=-C	A
FB	D=-D	A

```
FC          A=-A-1      A
FD          B=-B-1      A
FE          C=-C-1      A
FF          D=-D-1      A
```

Appendix D

System Calls

The Appendix lists the addresses of system entry points.

There is no warranty that a system call will work under any circumstance, thus we shall be not liable for incidental damages or loose of data consequential to the use of this documentation.

This material has been created using Smart Technology proprietary development system. In some cases may be there are some differences of interpretation respect to BBS listings.

Address	Name
02911	SysBin
02933	Real
02955	Lreal
02977	Complx
0299D	LCmplx
029BF	Char
029E8	Array
02A0A	Larray
02A2C	String
02A4E	BinInt
02A74	List
02A96	Dir
02AB8	Algbr
02ADA	Unit
02AFC	Tagged
02B1E	Grob
02B40	Lib
02B62	Backup
02B88	Libdat
02D9D	Prog
02DCC	Code
02E48	Global
02E6D	Local
02E92	Xlib
03019	=SKPNXT
0312B	End
0314C	Depth
03188	Dup
031AC	Dup2
031D9	DupN
03223	Swap
03244	Drop
03258	Drop2
0326E	DropN
03295	Rot
032C2	Over
032E2	Pick
03325	Roll
0339E	Rolld
03562	Count_Array_Elem
035A9	Arry→{<>[<>]}

03A81	TRUE
03AC0	FALSE
03AF2	T/F_NOT
03B46	Drop_If_(2)=F
03B75	If(2)=T_Drop/Nip
03B97	Same
03C64	Obj→<Typ>_or_<0>
03CA6	T_If_h=0
03CC7	T_If_h#0
03CE4	T_If_h2<h1
03D19	T_If_h2=h1
03D4E	T_If_h2#h1
03D83	T_If_h2>h1
03DBC	h=h2+h1
03DE0	h=h2-h1
03DEF	h=h+1
03E0E	h=h-1
03E2D	h=h+2
03E4E	h=h-2
03E6F	h=h*2
03E8E	h=h/2
03EB1	h=h2_AND_h1
03EC2	h=h2*h1
03EF7	h2/h1→<r>_<q>
03F8B	<Real>
03F95	<Cmplx>
03F9F	<List>
03FA9	<Global>
03FB3	<Rpl>
03FBD	<Alg>
03FC7	<Dir>
03FD1	<Local>
03FDB	<Sysbin>
03FE5	<Unit>
03FEF	<0>
03FF9	<1>
04003	<2>
400D	<3>
04017	<4>
04021	<5>
0402B	<6>
04035	<7>
0403F	<8>
04049	<9>
04053	<A>
0405D	<B>
04067	<C>
04071	<D>
0407B	<E>
04085	<F>
0408F	<10>
04099	<11>
040A3	<12>
040AD	<13>
040B7	<14>
040C1	<15>
040CB	<16>
040D5	<17>
040DF	<18>
040E9	<19>
040F3	<1A>
040FD	<1B>
04107	<1C>
04111	<1D>
0411B	<1E>
04125	<1F>
0412F	<20>
04139	<21>
04143	<22>
0414D	<23>
04157	<24>
04161	<25>
0416B	<26>
04175	<27>
0417F	<28>
04189	<29>
04193	<2A>
0419D	<2B>
04D64	<Errn>→Msg$
04DD7	<lllnn>→<nn><ll>
05023	=DOERRA
05089	First_Obj
050ED	$→Char
05143	=RS&CNT
05153	Cdr
0516C	Cdr$
05176	<FFFFF>
05193	$_Append_$
0521F	Append_list
0525B	$_Add_Char
052EE	$_Append_Char
052FA	Append_list
05331	→Comp
05459	→List

0546D	<n>→Alg
054AF	Comp→
0556F	Check_$=""
055B7	T_If_Empty_List
055DF	""
055E9	{}
05636	Len($)
0567B	List_Size
056B6	Get_Elem_&_Addr
05733	<><>$Sub
05821	<><>Lsub
05944	Niblen_&_Cksm
05A03	#→<h>
05A75	<h>→Char
05BE9	'NAME'→"NAME"
05C27	R→c
05D2C	C→r
05EC7	Tag→
05F42	Garbage_Collect
05F61	Mem
06529	=PSR1R0
06537	=PUSHR0
06641	=POP<>A
06657	Newob
0679B	=SAVPTR
067D2	=RSTPTR
06E8E	Cont_RPL
06E97	Push_Nxt_Addr
06F8E	Eval_Thread_only
06FD1	Eval_Next_Rtn
0712A	If_T_Then_Skip
0714D	Skip
071A2	Loop
071E5	End_Loop
071EE	If_F_Sk2_ExitLp
07221	Counter
07334	Next
073C3	Start_0→(1)-1
073CE	Start_1→(1)-1
073DB	Start_1→(1)
073F7	Start_(1)→(2)-1
07497	Destroy_Locals
074D0	Sto_Locals
074E4	Sto_Pairs_Local
07D27	Sto_Local
0811C	Get_<MsgTbl>_T/F
08D92	Home
0BBED	TRUE_TRUE
0F33A	→Unit
0F34E	Unit→
0F5FC	Abs_U
0F615	U→-u
0F6A2	Unit+
0FCFA	Int_U
0FD22	Floor_U
0FD36	Ceil_U
10F86	Doerr_Syntax
11679	Dsp_Grob
11CF3	→3_Grob
11D00	→2_Grob
11F80	→1_Grob
128B0	Put_Grob
1314D	Show_Stack
1400E	Clearerr
14039	Get_<Errn>
14065	Errm$
14088	→Str
140AB	Disp
140F1	Chr$
1410F	Num
14137	Str→
1415A	Beep
1420A	$>$
142A6	$<$
142BA	$>=$
142E2	$<=$
142FB	Freeze
15007	Doerr_n
1501B	Doerr_#
1502F	Doerr_<>
15048	Doerr_$
1578D	→Num
1592D	LstTok=0_&_CkD>0
15B31	Obj→String
166E3	Fix
166EF	Sci
166FB	Eng
16707	Std
167BF	T_If_Flag_51_off
18513	Sto_Global
1854F	Purge_Global
18779	Vars

1884D	Last_Tok=0
18873	$_and_$
18887	$_or_$
1889B	$_xor_$
188D2	Not_$
18A1E	No_Args
18A5B	Check_3
18A68	Check_Depth_>=_3
18A80	Check_2
18A8D	Check_Depth_>=_2
18AA5	Check_1
18AB2	Check_Depth_>=_1
18B6D	Check_5
18B7A	Check_Depth_>=_5
18B92	Check_4
18B9F	Check_Depth_>=_4
18C34	Check_N
18C4A	Check_Depth_>=_N
18C74	=SVLTOK
18C92	Doerr_Undef_Name
18CA2	Doerr_Bad_Arg_V
18CB2	Doerr_Bad_Arg_T
18CC2	Doerr_Too_Few
18CCE	=DOERRC
18CD7	R→ABS<>
18CEA	R→<h>
18D07	=PSACNT
18DBF	<h>→R
18EBA	Eval_comp
18ECE	Select_1
18EDF	Select_2
18EF0	Select_3
18F01	Select_4
18F12	Select_5
18F9D	Case_Type
18FB2	Check_1_Type
1945C	<Prlg>_Cklist
194F7	R2_R1→h2_h1
1950B	<h2>_<h1>→R2_R1
19529	{<>[<>]}→{R[R]}
1957B	ASR
1959B	RL
195BB	RLB
195DB	RR
195FB	RRB
1961B	SL
1963B	SLB
1965B	SR
1967B	SRB
1969B	R→B
196BB	B→R
1974F	→UNIT
198FE	STOALRM
19928	RCLALRM
19948	FNDALRM
19972	DELALRM
19992	TSTR
1A105	CRDIR
1A125	PATH
1A140	HOME
1A15B	UPDIR
1A16F	Updir
1A194	VARS
1A1AF	TVARS
1A1D9	BYTES
1A1FC	Obj_Bytes&Cksm
1A265	Name_Bytes&Cksm
1A2BC	NEWOB
1A2DA	T_If_Rom_Obj
1A339	DOERR
1A36D	ERR0
1A388	ERRN
1A3A3	ERRM
1A3BE	EVAL
1A3FE	IFTE
1A4A3	Ifte
1A4CD	IFT
1A4F0	Ift
1A513	s_Ift
1A52E	SYSEVAL
1A547	Syseval
1A584	DISP
1A5A4	FREEZE
1A5C4	BEEP
1A5E4	→NUM
1A604	LASTARG
1A631	Lastarg
1A71F	WAIT
1A738	Wait
1A7B5	Time_Wait
1A858	CLLCD
1A873	KEY
1A8D8	=
1A995	NEG

1AA1F	ABS
1AABD	pi
1AADF	MAXR
1AB67	+
1AC93	Ins_list
1ACA7	$+R
1ACBB	R+$
1AD09	-
1ADEE	*
1AF05	/
1B02D	^
1B124	R^r
1B278	INV
1B4AC	SIN
1B505	COS
1B55E	TAN
1B5B7	SINH
1B606	COSH
1B655	TANH
1B6A4	ASIN
1B72F	ACOS
1B79C	ATAN
1B7EB	ASINH
1B830	ACOSH
1B8A2	ATANH
1B905	EXP
1B94F	LN
1B9C6	LOG
1BA3D	ALOG
1BA8C	LNP1
1BAC2	EXPM
1BB02	!
1BB41	FACT
1BB6D	IP
1BBA3	FP
1BBD9	FLOOR
1BC0F	CEIL
1BC45	XPON
1BC71	MAX
1BCE3	MIN
1BD55	RND
1BDD1	TRNC
1BE4D	MOD
1BE9C	MANT
1BFDE	DET
1C274	SF
1C2D5	CF
1C313	FS?
1C32C	?fs
1C360	FC?
1C379	?fc
1C399	DEG
1C3B4	RAD
1C3CF	GRAD
1C3EA	FIX
1C403	Fix
1C41E	SCI
1C437	Sci
1C452	ENG
1C46B	Eng
1C486	STD
1C4A1	FS?C
1C4BA	Fs?c
1C520	FC?C
1C539	Fc?c
1C559	BIN
1C574	DEC
1C5C5	STWS
1C5FE	RCWS
1C619	RCLF
1C637	Sysflag→#
1C64E	Usrflag→#
1C67F	STOF
1C783	→LIST
1C79E	R→C
1C7CA	RE
1C819	IM
1C85C	SUB
1C8BB	$_Sub
1C8CF	List_Sub
1C8EA	REPL
1C95A	LIST→
1C973	List→
1C9B8	SIZE
1CA26	$size
1CA3A	List_Size_R
1CA4E	Array_Dims
1CA62	Grob_Size
1CA85	Pict_Size
1CAB4	POS
1CAD7	Pos_$
1CAF0	Pos_List
1CB0B	→STR
1CB26	STR→

1CB46	NUM
1CB66	CHR
1CB86	TYPE
1CB90	Type
1CDB1	Type_of_Array
1CDD4	Type_of_RPL
1CE28	VTYPE
1CEE3	EQ→
1CF2E	Eq→
1CF7B	OBJ→
1CFD0	Alg→
1D009	→ARRY
1D02C	n→Arry
1D040	{}→Arry
1D092	ARRY→
1D0AB	Arry→
1D0DF	RDM
1D186	CON
1D2DC	IDN
1D392	TRN
1D407	PUT
1D484	Glob_put
1D4DE	Array_put
1D524	List_put
1D565	Loc_put
1D5DF	PUTI
1D65C	Glob_puti
1D6B6	Arry_puti
1D701	List_puti
1D747	Loc_puti
1D7C6	GET
1D825	Name_get
1D86B	Array_get
1D898	List_get
1D8C7	GETI
1D926	Name_geti
1D96C	Arry_geti
1D9BC	List_geti
1DB5B	Chk_for_Get_Args
1DC00	Put_Obj
1DD06	V→
1DE66	→V2
1DEC2	→V3
1E07E	PMIN
1E09E	PMAX
1E0BE	AXES
1E126	RES
1E25F	ERASE
1E27A	PX→C
1E29A	C→PX
1E2F0	PVIEW
1E31A	PIXON
1E344	PIXOFF
1E36E	PIX?
1E3EC	BOX
1E416	BLANK
1E436	PICT
1E456	GOR
1E4E4	GXOR
1E572	LCD→
1E58D	→LCD
1E5AD	→GROB
1E606	TEXT
1E761	SAME
1E783	AND
1E7DD	R_and_R
1E809	OR
1E863	R_or_R
1E88F	NOT
1E8D9	Not
1E8F6	XOR
1E946	R_xor_R
1E972	==
1EA30	Any==Any
1EA44	Tag==any
1EA6C	R==c
1EA76	C==r
1EA9D	#
1EB51	Any#Any
1EB65	Tag#Any
1EB8D	R#c
1EB97	C#r
1EBBE	<
1EC40	R<r
1EC5D	>
1ECDF	R>r
1ECFC	<=
1ED7E	R<=r
1ED9B	>=
1EE1D	R>=r
1EEA4	CR
1F1D4	integral
1F201	Integral

1F500	QUOTE
1F542	Quote(f)
1F55D	APPLY
1F9C4	→Q
1F9E9	→Qpi
1FA07	→Fraction
1FB87	DUP
1FBA2	DUP2
1FBBD	SWAP
1FBD8	DROP
1FBF3	DROP2
1FC0E	ROT
1FC29	OVER
1FC44	DEPTH
1FC64	DROPN
1FC7F	DUPN
1FC9A	PICK
1FCB5	ROLL
1FCD0	ROLLD
1FCEB	CLEAR
1FD0B	STOsigma
1FD2B	CLsigma
1FD46	RCLsigma
1FD61	sigma+
1FD8B	sigma-
1FDA6	Nsigma
1FDC1	CORR
1FDDC	COV
1FDF7	sigmaX
1FE12	sigmaY
1FE2D	sigmaX^2
1FE48	sigmaY^2
1FE63	sigmaX*Y
1FE7E	MAXsigma
1FE99	MEAN
1FEB4	MINsigma
1FECF	SDEV
1FEEA	TOT
1FF05	VAR
1FF20	LR
1FF7A	PREDV
1FF9A	PREDY
1FFBA	PREDX
1FFDA	XCOL
1FFFA	YCOL
2001A	UTPC
2003A	UTPN
2005A	UTPF
2007A	UTPT
2009A	COLsigma
200C4	SCLsigma
200F3	sigmaLINE
2010E	BINS
20133	BARPLOT
20167	HISTPLOT
2018C	SCTRPLT
201B1	LINFIT
201D6	LOGFIT
201FB	EXPFIT
20220	PWRFIT
2025E	BESTFIT
202CE	SINV
2034D	SNEG
203CC	SCONJ
2044B	STO+
20538	STO-
2060C	STO/
20753	STO*
208F4	INCR
209AA	DECR
20A15	COLCT
20A49	EXPAN
20B40	RCL
20B81	Rcl_name
20B9A	Rcl_bypth
20CAD	Rcl_Pict
20CCD	STO
20D65	DEFINE
20EFE	PURGE
20F35	Purge_{ }
20F8A	Purge_PICT
20FAA	MEM
20FD9	ORDER
20FF2	Order
210FC	CLVAR
2115D	TMENU
21196	MENU
211E1	RCLMENU
211FC	PVARS
2123A	PGDIR
2137F	MERGE
213D1	FREE
2142D	LIBS
21448	ATTACH

21461	Attach
2147C	DETACH
21495	Detach
214A9	Check_valid_LID
214F4	Sto_Port
215BF	Sto_Backup/Lib
21660	Nip_TRUE
21761	Rcl_port
217C7	Eval_Tag
217F1	Purge_Tagged
21C6F	<>Attach
21CE5	<LID>_Detach
21E75	XMIT
21E95	SRECV
21EB5	OPENIO
21ED5	CLOSEIO
21EF0	SEND
21F24	KGET
21F62	RECN
21F96	RECV
21FB6	FINISH
21FD1	SERVER
21FEC	CKSM
2200C	BAUD
2202C	PARITY
2204C	TRANSIO
2206C	KERRM
22087	BUFLEN
220A2	STIME
220C2	SBRK
220DD	PKT
224CA	INPUT
224F4	ASN
22514	STOKEYS
22548	DELKEYS
22586	RCLKEYS
225BE	→TAG
22633	DTAG
22EC3	IF
22EFA	THEN
22F22	Then
22F4F	s_THEN
22FB5	ELSE
22FD5	END_IF
22FEB	→
23033	WHILE
2305D	REPEAT
230C3	DO
230ED	UNTIL
23103	START
23144	r_r_Start
23167	r_s_Start
23180	s_r_Start
231A0	FOR
231E1	r_r_For
23213	s_s_For
2322C	s_r_For
2324C	NEXT
23380	STEP
233A8	s_Step
233C1	_Step
233DF	IFERR
23472	HALT
234C1	→
235FE	>>_local
2361E	<<
23639	>>
23654	'
23679	_'
23694	END_WHILE
236B9	END_DO
2372E	'stop
2373F	'noname
23754	{start}
23768	If
2378D	CASE
237A8	THEN
23824	PROMPT
23879	{'ioinprogress}
238A4	Parse_String
23989	{}
239CF	5_Drop_Nip_True
25D3A	#1111 1
26A2D	T_If_Function
29FDA	=POP1R+
2A2B4	0
2A2C9	1
2A2DE	2
2A2F3	3
2A308	4
2A31D	5
2A332	6
2A347	7
2A35C	8

2A371	9
2A386	-1
2A39B	-2
2A3B0	-3
2A3C5	-4
2A3DA	-5
2A3EF	-6
2A404	-7
2A419	-8
2A42E	-9
2A443	pi
2A458	pi_Long
2A472	maxreal
2A487	-maxr
2A49C	minr
2A4B1	-minr
2A4C6	0_Long
2A4E0	1_Long
2A4FA	2_Long
2A514	3_Long
2A52E	4_Long
2A548	5_Long
2A562	.1_Long
2A57C	.5_Long
2A596	10_Long
2A5B0	LongReal→R
2A5C1	Real→LongReal
2A76B	Not_R
2A799	T_If_>0
2A7CF	F_If_0
2A900	Abs
2A920	R→-R
2A930	Mant
2A974	R=R2+R1
2A981	R=R2-R1
2A9BC	R=R2*R1
2A9FE	R=R2/R1
2AAAF	R=1/R
2ABDC	Mod
2AF60	Int
2AF73	Ceil
2AF86	Floor
2D9F5	Server
2E5AB	Name_Send
2E6EB	{}_Send
2E876	Finish
2E8D1	Pkt
2EC84	Baud
2ECCA	Parity
2ED10	Transio
2ED4C	Cksm
2EDA6	Kerrm
2EDE1	Buflen
2EDF5	Stime
2EE18	Sbrk
2EE6F	Xmit
2EE97	Srecv
315C6	Closeio
34D2B	{"}
34D30	"
3558E	<>_Arry_Read
35DEB	Neg_Arry
35F8F	[]→[rept]
35FEE	[]→[impt]
36039	[R]→[C]
36278	Arry-Arry
369CB	Abs_[]
36A2A	Det
37B44	Newob_If_Needed
37BCB	For_1→Size([])
3922F	Sleep
3A1FC	Upd_Menu
3A4CE	$_<h>_<8-h>_Disp
3A7F3	[ENTER]
415C9	Rclmenu
41679	n_Tmenu
41B28	Asn
41F65	KeyWait→<#k><f>
42F44	Editor
43395	$_$_Input
433CC	$_{}_Input
4B60C	Erase
4F0AC	Px→c
4F179	C→px
4F37C	Sto_Pict
4F3D1	#h2#h1→<h2><h1>
4F3EF	C→Pixon
4F458	L→Pixon
4F471	C→Pixoff
4F48A	L→Pixoff
4F4A3	C→Pix?
4F4BC	L→Pix?
4F665	Px_Box

4F688	C_Box
4F6A1	Blank
4F8D1	Grob+grob
4F999	Gr_L_Repl
4F9F3	Gr_C_Repl
4FA2F	Pict_Repl
4FA7A	Repl_list
4FAF7	repl$
4FB74	Gr_L_sub
4FBC4	Gr_C_sub
4FBF6	Pict_sub
4FC28	Inverse
4FC3C	Pict_invr
503C5	Text
50438	→Lcd
5046A	Cllcd
5048D	→Grob
51532	Px→2<>
5198F	Drop+_0
519A3	C→rept
519B7	C→impt
51B70	C→-C
51BD0	C=C2+R1
51BF8	C=R2+C1
51C16	C=C2+C1
51CD4	C=R2-C1
51CE8	C=C2-R1
51CFC	C=C2-C1
51D4C	C=C2*R1
51D60	C=R2*C1
51D88	C=C2*C1
51E19	C=R2/C1
51E64	C=C2/R1
51EC8	C=C2/C1
51EFA	C=1/C
52062	Abs_C
52342	C=R2^C1
52360	C=C2^R1
52374	C=C2^C1
52D26	{"_"_"_"}
53784	T_If_<Flag>_Set
5380E	T/F→1/0
53D04	b=b2_AND_b1
53D15	b=b2_OR_b1
53D26	b=b2_XOR_b1
53D4E	b=NOT_b
53D5E	Sl
53D6E	Slb
53D81	Sr
53D91	Srb
53DA4	Rr
53DE1	Rrb
53E0C	Rl
53E3B	Rlb
53E65	Asr
53EA0	b=b2+b1
53EB0	b=b2-b1
53EC3	b=-b
53ED3	b=b2*b1
53F05	b=b2/b1
5429F	b=R2/b1
542BD	b=b2/R1
542D1	b=R2*b1
542EA	b=b2*R1
542FE	b=R2-b1
5431C	b=b2-R1
54330	b=R2+b1
54349	b=b2+R1
5435D	b→R
543F9	R→b
544D9	b2==b1
544EC	b2#b1
54500	b2>b1
5452C	b2>=b1
5453F	b2<=b1
54552	b2<b1
54565	Ifte(x)
54D12	Maxr
54D35	Pi
54EA0	Symb→rept
54EB9	Symb→impt
54EEB	f→-f
54F04	Abs(f)
5518E	Int(f)
551C0	Floor(f)
551D9	Ceil(f)
5520B	Mant(f)
55927	R=r
55F5D	Symb+Symb
59F91	Alg_Len
5A60F	{'piflag}
5E370	Dup<n>
60EE7	ABC→BAC
60F0E	ABC→BA

60F21	ABC→BC
60F33	ABC→CBA
60F4B	3_Dropn
60F54	7_Dropn
60F66	6_Dropn
60F72	5_Drop
60F7E	4_Drop
60F9B	Nip
60FAC	ABC→CAB
60FBB	ABCD→BCDA
60FD8	5_Roll
61002	6_Roll
6103C	8_Roll
6106B	7_Roll
6112A	ABC→C
611FE	3_Pick
6121C	4_Pick
6123A	5_Pick
6125E	6_Pick
61282	7_Pick
612A9	8_Pick
61380	AB→BAB
613B6	Rcl_Last_Loc
613E7	Rcl_2-Last_Loc
61891	If#0_Dup_Skip
618D3	If_(1)=(2)_Th_El
6191F	If_T_D2_Do_Nx_Ex
6194B	If_T_Drp_Rtn
61970	If_T_Drop2_Exit
61993	If_T_Do_nxt_Rtn
619AD	If_F_Then_Nx_Rtn
619BC	If_T_Then
61A02	=Set_T
61A2C	If_T_Nx_Ex_el_Rt
61A3B	If_T_Rtn
61AD8	If_T_Then_Else
61FA9	T_If_Rom_Obj
620D9	=IFcT/F
62154	Dup_T_If_String
62169	T_If_Dup_Real
6223B	T_If_Real_Array
62266	<h>_T_If=<0>
6226F	=IFA=0T
62278	=GET<>A
622A7	T_If_<h>=1
622B6	T_If_<>_#_1
622C5	T_If_<>-1=<0>
622D4	T_If_<h>#0
622EF	SwapAdd$
624BA	Min(<h2>,<h1>)
624C6	Max(<h2>,<h1>)
62535	Push_<0>
6256A	<>+3
6257A	<>+4
6258A	<>+5
6259A	<>+6
625AA	<>+7
625BA	<>+8
625CA	<>+9
625DA	<>+10
625EA	<>+12
625FA	<>-3
6260A	<>-4
6261A	<>-5
6262A	<>-6
62636	=<>+C
62747	AB→BAA
6289B	<h2>_<h1>_T_If_<
62986	R_If_T_Nxt_el_Sk
629BC	If_T&T_Then
62B88	Comp→_Drop
62B9C	Get_Elem
62C7D	ABC→BCACA
62CA5	ABC→BCAC
62CE1	R→(<h>_<h>)
62D31	AB→AAB
62D59	""_Swap
62E3A	<0>_Swap
62E7B	R→<h>_Swap
62FB1	AB→BBA
630B5	ABC→ABCAC
6312D	10_Rolld
631B9	<2>→list
631E1	Dup_Comp→
63209	Dup_Chk_If_$=""
6321D	Dup_T_If_Empty_L
63231	Dup_Comp_Size
63353	If_<h>=1_Skip
63411	Dup_Loop_counter
63498	For_1→Comp_Size
634B6	Last_Rcl&Destroy
634F7	TRUE_FALSE
6351F	Push_<0>_FALSE

6364B	T_If_(2)=<0>
6365F	(2)&T_If_(1)<(2)
636A0	Comp→T_If_Size=1
6372C	<h2>_<h2+h1>
637A4	<h1>_<h2>-<h1>
6383A	<>_Doerr_noOwner
63A6F	Dup_T_If_=_{}
63AB0	Swap_<1>
63AC4	Push_<1>_<1>
63B05	If_T_Doerr_Bad
63B2D	Check_Real
63BAA	Dup_Not_R
63CFE	If_Same_Nx_el_Sk
63D12	If_h2<h1_Th_Else
63D3A	If_(2)#(1)_Th_El
63D4E	If_<>_>2_Then
63E48	If=<0>_Do_El_Skp
63E9D	If_<_Then_Else
644A3	Pos_List_<>
645B1	Pos($)
6475C	Char→$
64775	Dtag
647A2	Dtag_level_2
64B12	<2C>
64B1C	<2D>
64B26	<2E>
64B30	<2F>
64B3A	<30>
64B44	<31>
64B4E	<32>
64B58	<33>
64B62	<34>
64B6C	<35>
64B76	<36>
64B80	<37>
64B8A	<38>
64B94	<39>
64B9E	<3A>
64BA8	<3B>
64BB2	<3C>
64BBC	<3D>
64BC6	<3E>
64BD0	<3F>
64BDA	<40>
64BE4	<41>
64BEE	<42>
64BF8	<43>
64C02	<44>
64C0C	<45>
64C16	<46>
64C20	<4A>
64C2A	<4F>
64C34	<50>
64C3E	<51>
64C48	<52>
64C52	<53>
64C5C	<54>
64C66	<55>
64C70	<56>
64C7A	<57>
64C84	<5B>
64C8E	<60>
64C98	<61>
64CA2	<62>
64CAC	<64>
64CB6	<65>
64CC0	<6F>
64CCA	<70>
64CD4	<71>
64CDE	<72>
64CE8	<73>
64CF2	<74>
64CFC	<75>
64D06	<7A>
64D10	<80>
64D1A	<82>
64D24	<83>
64D2E	<8F>
64D38	<91>
64D42	<92>
64D4C	<9A>
64D56	<9E>
64D60	<9F>
64D6A	<A0>
64D74	<A1>
64D7E	<A2>
64D88	<A5>
64D92	<A6>
64D9C	<A7>
64DA6	<A9>
64DB0	<AA>
64DBA	<AE>
64DC4	<B1>
64DCE	<BB>

64DE2	<CC>
64DEC	<D0>
64DF6	<E1>
64E00	<EA>
64E0A	<EE>
64E14	<F0>
64E1E	<FD>
64E28	<FF>
64E32	<100>
64E3C	<102>
64E46	<106>
64E50	<107>
64E5A	<110>
64E64	<111>
64E6E	<123>
64E78	<124>
64E82	<131>
64E8C	<132>
64E96	<133>
64EA0	<134>
64EAA	<135>
64EB4	<136>
64EBE	<137>
64EC8	<138>
64ED2	<139>
64EDC	<13A>
64EE6	<13B>
64EF0	<13D>
64EFA	<13E>
64F04	<151>
64F0E	<200>
64F18	<205>
64F22	<311>
64F2C	<411>
64F36	<412>
64F40	<444>
64F4A	<451>
64F54	<452>
64F5E	<510>
64F68	<511>
64F72	<550>
64F7C	<610>
64F86	<650>
64F90	<700>
64F9A	<861>
64FA4	<862>
64FAE	<865>
64FB8	<86E>
64FC2	<A03>
64FCC	<A11>
64FD6	<A12>
64FE0	<A1A>
64FEA	<A21>
64FF4	<A22>
64FFE	<A2A>
65008	<A61>
65012	<A62>
6501C	<A65>
65026	<A6E>
65030	<AA1>
6503A	<AA2>
65044	<AAA>
6504E	<C06>
65058	<C07>
65062	<C08>
6506C	<C0A>
65076	<C0B>
65080	<DFF>
6508A	<E00>
65094	<70000>
6509E	<FFFFF>
650A8	e
650BD	.5
650D2	-.5
650E7	10
650FC	180
65111	200
65126	360
6513B	400
65150	"]"
6515C	"["
6516A	"["
65176	"{"
65182	"}"
6518E	"#"
6519A	" _"
651A6	"$"
651B2	"&"
651BE	LF$
651CA	">>"
651D6	"<<"
651E2	"E"
651FA	"sigma"
65206	"|"

65212	14_blank$
65238	CR$
65244	"der"
65254	" "
65260	"UNKNOWN"
65278	""""
65284	""
65290	","
6529C	"."
652A8	";"
652B4	"("
652C0	")"
652CC	"^"
652D8	"*"
652E4	"/"
652F0	"+"
652FC	"-"
65308	"="
65314	"root"
65320	"delta"
6532C	"GROB"
6533E	"C$"
6534C	"0"
65358	"1"
65364	"2"
65370	"3"
6537C	"4"
65388	"5"
65394	"6"
653A0	"7"
653AC	"8"
653B8	"9"
653C4	<726A5>
653CE	<72704>
653D8	<72DCF>
653E2	<72F1E>
653EC	<736F9>
653F6	<7232C>
65400	<7260A>
6540A	<72281>
65414	<72FE6>
6541E	char_0
65425	char_31
6542C	char_34
65433	char_35
6543A	char_42
65441	char_43
65448	char_44
6544F	char_45
65456	char_46
6545D	char_47
65464	char_48
6546B	char_49
65472	char_50
65479	char_51
65480	char_52
65487	char_53
6548E	char_54
65495	char_55
6549C	char_56
654A3	char_57
654AA	char_58
654B1	char_59
654B8	char_60
654BF	char_61
654C6	char_62
654CD	char_65
654D4	char_66
654DB	char_67
654E2	char_68
654E9	char_69
654F0	char_70
654F7	char_71
654FE	char_72
65505	char_73
6550C	char_74
65513	char_75
6551A	char_76
65521	char_77
65528	char_78
6552F	char_79
65536	char_80
6553D	char_81
65544	char_82
6554B	char_83
65552	char_84
65559	char_85
65560	char_86
65567	char_87
6556E	char_88
65575	char_89
6557C	char_90
65583	char_97
6558A	char_98

65591	char_99
65598	char_100
6559F	char_101
655A6	char_102
655AD	char_103
655B4	char_104
655BB	char_105
655C2	char_106
655C9	char_107
655D0	char_108
655D7	char_109
655DE	char_110
655E5	char_111
655EC	char_112
655F3	char_113
655FA	char_114
65601	char_115
65608	char_116
6560F	char_117
65616	char_118
6561D	char_119
65624	char_120
6562B	char_121
65632	char_122
65639	char_141
65640	char_171
65647	char_187
6564E	char_128
65655	char_136
6565C	char_132
65663	char_40
6566A	char_10
65671	char_135
65678	char_41
6567F	char_133
65686	char_32
6568D	char_95
65694	char_91
6569B	char_93
656A2	char_123
656A9	char_125
656B0	char_137
656B7	char_138
656BE	char_139
656E5	"XYZ"
656F5	"<<>>"
65703	"{}"
65711	"[]"
6571F	"''"
6572D	"::"
6573B	"()"
65749	""
65757	"ECHO"
65769	"EXIT"
6577B	Undef$
65796	"RAD"
657A6	"GRAD"
70000	RAMST
704EA	KEYBUF
70551	MENUGR
70556	STKGRO
70579	SAV-D1
705B0	SAV-D0
706C5	SYSFLG
706D5	USRFLG

Appendix E

Error Messages

The following error messages are unique to the SmartROM:

Error number	Error Message
33501	Conformability
33502	Type Mismatch
33503	Invalid Sub-List
33504	Argument Out Of Range
33505	KEYS/ACTIONS mismatch
33506	Missing Var

Conformability

Issued when two matrices are not compatible for row-by-column multiplication. Given a matrix A(m,n), the second must have the form B(n,p).

Type Mismatch

Issued when a meta-object contains objects of different types. This restriction is checked by some commands when type homogeneity is required. A typical example is SRT.

Invalid Sub-List

Issued when a list-matrix contains rows of different size.

Subscript Out Of Range

Issued when a subscript in a list-matrix overflows or underflows its valid range.

KEYS/ACTIONS mismatch

Issued when the length of the lists KEYS and ACTIONS are not compatible. Length(ACTIONS) := Length(KEYS)+1

Missing Var

Issued when the list KEYS or the list ACTIONS are missing. Issued by the application →FONT when the global name FONT is missing or cannot be found in the search path or does not contain a valid data structure for the operation being performed.

Appendix H

Hidden Commands

This appendix contains the complete list of SmartROM hidden commands arranged in logic categories for easier referencing. Each command along with its entry and exit conditions is discussed in detail in the SmartROM *Hidden Commands Reference*.

	Original name	XLIB number
String Utilities	center$	244
	find$	156
	lfpos	186
	lines→	142
	→lines	98
	ltrim$	225
	lwc$	224
	mcentr$	245
	member$	214
	→msg$	220
	norm$	222
	null	125
	replace$	119
	revs	218
	rowcol	115
	rpt	114
	rtrim$	226
	span$	215
	split$	248
	splith	105
	trim$	221
	upc$	223
Binary string Utilities	addp	230
	expbuf	243
	null	125
	popp	231
	revb	219
	rpt	114
System Binary Utilities	log2	141
	ord	232

	revsys	203
	todd	117
List Manipulation Utilities	bind	191
	checkl	190
	chl?	170
	ckl2r	188
	delcol	163
	delrow	161
	diff	240
	findobj	153
	getcol	200
	idx?	146
	inter	239
	l2m	144
	lget	143
	lop1	140
	lopn	139
	lput	138
	lvop	137
	nget	206
	npos	212
	nput	207
	null	125
	putobj	208
	red	241
	replace	120
	rpt	114
	splitl	104
	splito	249
	union	238
General purpose Utilities	#k	194
	ckr	197
	ckrol	196
	getaddr	210
	keywait	145
	ncount	211
	rptcmd	113
Program editing utilities	findobj	153
	#k→$	195
	nget	206
	npos	212
	nput	207
	putobj	208
	replace	120
	search	107
	splitl	104
Stack Manipulation Utilities	c2m	171
	hdrop	150
	hdup	149
	hshift	148
	mark	136
	mds	185
	mus	182

	gaddup	151
	gop	235
	line	234
	linetypes	237
	patterns	229
	polygon	247
	ppardef	246
	rect	233
	rpt	114
	scan	213
	scanp	242
	vfill	227
Object Manipulation Utilities	apply	205
	chl?	170
	chset?	169
	chst?	168
	dot	116
	findobj	153
	nget	206
	npos	212
	nput	207
	null	125
	putobj	208
	replace	120
	rpt	114
	tifc	209

Table of Contents