META KERNEL

Manual


[Dec. 1998 - Cut-down version]
Credits
Jean-Yves Avenard
Cyrille de Brebisson
Christian Bourgeois
Etienne de Foras
Gérald Squelart

Thanks to
Jake Schwartz (without whom this English manual wouldn't exist!)
Paul Courbis (the 'Voyage' is our bible)
All at Maubert Electronic
Hewlett Packard (for the HP48 and the RPLMAN doc)

And in no particular order (alphabetical)
Victor Achiaga
Al Arduengo
Jonathan Aidan
Stephane Albi
Sébastien Casartelli
Jorge Costa
Stephane Doat
Damien Harper
Joe Horn
Stephen Janin
Christian Meland
Bernard Parisse

© 1996-1998 MDG Corp.
No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of MDG Corp.
Printed in France.
## Contents

** USING THE META KERNEL .......................................................................................................... 9  
1 INSTALLATION ......................................................................................................................... 9  
2 GENERAL PRESENTATION ......................................................................................................... 9  
3 STACK DISPLAY ....................................................................................................................... 9  
4 CONFIGURATION .................................................................................................................... 10  
5 INTERACTIVE STACK ............................................................................................................... 13  
6 COMMAND LINE .................................................................................................................. 13  
7 COMMAND LIBRARY ............................................................................................................. 16  
8 EQUATION EDITOR ................................................................................................................. 17  
9 MATRIX EDITOR ..................................................................................................................... 23  
10 GROB EDITOR (PICTURE2) ..................................................................................................... 24  
11 FILER .................................................................................................................................... 27  

** UTILITIES ............................................................................................................................. 31  
1 FONT EDITOR (EDF) ............................................................................................................... 31  
2 MACHINE LANGUAGE COMPILER (MASD) ........................................................................... 32  
3 DISASSEMBLER .................................................................................................................... 43  
4 MISCELLANEOUS UTILITIES .................................................................................................. 43  

** APPENDIXES .......................................................................................................................... 48  
1 INTRODUCTION TO ASSEMBLY LANGUAGE ........................................................................ 48  
2 INTRODUCTION TO SYSTEM RPL ....................................................................................... 48  
3 TIPS AND TRICKS .................................................................................................................. 48  
4 THE MK FOR PROGRAMMERS ............................................................................................... 50  
5 METAKERNEL 3.0 : PREVIEW .................................................................................................. 61
# Table of Contents

## USING THE META KERNEL

1. INSTALLATION .................................................................................................................. 9
2. GENERAL PRESENTATION ................................................................................................. 9
3. STACK DISPLAY .................................................................................................................. 9
   3.1 Display ............................................................................................................................ 9
      3.1.1 Status area ................................................................................................................. 9
      3.1.2 Stack ......................................................................................................................... 9
      3.1.3 Menu labels .............................................................................................................. 10
   3.2 Display configuration ..................................................................................................... 10
      3.2.1 Changing the display font ....................................................................................... 10
      3.2.2 Modifying the current font ..................................................................................... 10
      3.2.3 Changing the mini font ......................................................................................... 10
4. CONFIGURATION ............................................................................................................... 10
   4.1 MK variables ................................................................................................................ 11
   4.2 MK user and system flags (MKUF, MKSF) ..................................................................... 11
5. INTERACTIVE STACK ......................................................................................................... 13
6. COMMAND LINE ............................................................................................................... 13
   6.1 Generalities ................................................................................................................... 13
   6.2 Editing menu ................................................................................................................ 13
      6.2.1 Major options .......................................................................................................... 13
      6.2.2 Operations on the selection .................................................................................. 14
      6.2.3 Search operations .................................................................................................. 14
      6.2.4 Miscellaneous ...................................................................................................... 14
   6.3 Styles ............................................................................................................................ 14
      6.3.1 Generalities ........................................................................................................... 14
      6.3.2 Style menu ............................................................................................................ 14
      6.3.3 Using styles ......................................................................................................... 14
      6.3.4 Using fonts ........................................................................................................... 14
   6.4 Miscellaneous command line options ........................................................................... 15
      6.4.1 Full screen mode .................................................................................................. 15
      6.4.2 Auto-indent mode ............................................................................................... 15
      6.4.3 Special keystrokes for System RPL ................................................................. 16
      6.4.4 Big strings editing .............................................................................................. 16
   6.5 Shortcuts ...................................................................................................................... 16
7. COMMAND LIBRARY ........................................................................................................ 16
8. EQUATION EDITOR ......................................................................................................... 17
   8.1 First view of EQW ...................................................................................................... 17
      8.1.1 Introduction ............................................................................................................ 17
      8.1.2 Modes .................................................................................................................... 17
      8.1.3 Editing .................................................................................................................... 17
   8.2 Creating equations ....................................................................................................... 18
      8.2.1 Running EQW ....................................................................................................... 18
      8.2.2 Exiting EQW ......................................................................................................... 18
      8.2.3 Numbers and names ............................................................................................. 18
      8.2.4 Addition, subtraction, and multiplication ............................................................ 18
      8.2.5 Complex numbers and expressions ..................................................................... 18
      8.2.6 Moving in the equation ....................................................................................... 18
      8.2.7 Divisions .............................................................................................................. 18
      8.2.8 Exponents ............................................................................................................ 18
      8.2.9 Square root ........................................................................................................... 18
      8.2.10 Nth square root ................................................................................................. 19
      8.2.11 Parenthesized-argument functions ................................................................. 19
      8.2.12 User functions ................................................................................................ 19
      8.2.13 Parenthesized terms ......................................................................................... 19
      8.2.14 Differentiation .................................................................................................. 19
      8.2.15 Integration ......................................................................................................... 19
      8.2.16 Summations ...................................................................................................... 19
      8.2.17 Where function ............................................................................................... 19
   8.3 Manipulating equations ............................................................................................... 19
      8.3.1 Modifying equations with EQW ................................................................. 19
      8.3.2 Modifications ...................................................................................................... 19
      8.3.3 Temporary exit to the stack .............................................................................. 19
      8.3.4 Copy paste functions ......................................................................................... 19
      8.3.5 Deletions .............................................................................................................. 20
1 FONT EDITOR (EDF) ................................................................. 31
  1.1 Font editing ................................................................. 31
  1.2 CHARS mode .............................................................. 31
  1.3 A session with CHARS ................................................. 31
2 MACHINE LANGUAGE COMPILER (MASD)........................ 32
  2.1 Generalities on ML (Machine Language) .................... 32
  2.2 Launching Masd .......................................................... 32
  2.3 Generalities on Masd Syntax ....................................... 32
  2.4 Links ............................................................................ 32
  2.5 Using labels ................................................................. 32
  2.6 Using constants .......................................................... 33
  2.7 Expressions ................................................................. 33
  2.8 Skips ............................................................................ 34
  2.9 Macros ......................................................................... 35
  2.10 Filename conventions ................................................ 35
  2.11 Units .......................................................................... 35
  2.12 Library creation ........................................................ 35
  2.13 SysRPL mode ............................................................. 36
    2.13.1 Instructions .......................................................... 36
      a) Constants ............................................................... 36
      b) External (in entry points table) ............................... 36
      c) Tokens .................................................................... 36
      d) Decimal value (System Binary) .............................. 37
      e) Unnamed local variables ...................................... 37
    2.13.2 Special keystrokes in the command line ............ 37
    2.13.3 Program example ............................................... 38
  2.14 SATURN instructions syntax .................................... 38

3 REFERENCE ........................................................................... 38
  3.1 Generalities on the reference ...................................... 38
  3.2 References .................................................................... 38

4 EXAMPLES ............................................................................ 40
  4.1 Examples ................................................................. 40
  4.2 Common routines ....................................................... 40

5 ATRIX EDITOR ....................................................................... 41
  5.1 Generalities on the Atrix Editor ................................... 41
  5.2 Launching Atrix .......................................................... 41
  5.3 Instructions .................................................................... 41
  5.4 Data entry modes ....................................................... 42
  5.5 Graphs ........................................................................... 42
  5.6 Postscript ................................................................. 42
  5.7 Programming ............................................................ 42
  5.8 Functions ...................................................................... 42
  5.9 Sine ............................................................................. 43
  5.10 Numerical integration ............................................... 43
  5.11 FFT and DFT ........................................................... 43
  5.12 More general applications ....................................... 43
  5.13 Kaleidoscope ............................................................. 44
  5.14 Special routines ....................................................... 44

6 NUMERICAL PROBLEMS ................................................... 45
  6.1 Linear systems .......................................................... 45
  6.2 Initial value problems ............................................... 45
  6.3 Boundary value problems ......................................... 45
  6.4 Optimization ............................................................ 46

7 SIMULATIONS ....................................................................... 47
  7.1 Generalities on the simulations ................................... 47
  7.2 Examples ................................................................. 47

8 COMPUTER ORGANIZATION .............................................. 49
  8.1 Generalities on the computer organization .................. 49
  8.2 Microprocessor .......................................................... 49
  8.3 Memory ................................................................. 49
  8.4 Input-output ............................................................. 49

9 MATRIX EDITOR .................................................................... 50
  9.1 Presentation ............................................................. 50
  9.2 Usage .......................................................................... 50
  9.3 Reference ...................................................................... 50
  9.4 Keys ............................................................................ 50
  9.5 Expression entry ....................................................... 50
  9.6 Multidimensional matrices ....................................... 50
  9.7 Programming .......................................................... 50

10 GRIB EDITOR (PICTURE2) ................................................ 52
  10.1 Usage ........................................................................... 52
  10.2 Reference .................................................................... 52
  10.3 Commands .............................................................. 52
  10.4 ALPHA keys ............................................................ 53

11 FILER ................................................................................... 54
  11.1 Ports and directories screen ...................................... 54
  11.2 File selection screen ................................................ 54
  11.3 Menu .......................................................................... 54
  11.4 HEXA Editor ............................................................ 54
  11.4.1 Menu ................................................................. 54
  11.4.2 Keys ...................................................................... 55
  11.4.3 Edit mode .......................................................... 55

UTILITIES .................................................................................. 56
  1 FONT EDITOR (EDF) ................................................................. 56
  2 MACHINE LANGUAGE COMPILER (MASD).................... 56
  3 REFERENCE ........................................................................... 57
  4 EXAMPLES ............................................................................ 58
  5 ATRIX EDITOR ....................................................................... 59
  6 NUMERICAL PROBLEMS ................................................... 60
  7 SIMULATIONS ....................................................................... 61
  8 COMPUTER ORGANIZATION .............................................. 62
  9 MATRIX EDITOR .................................................................... 63
  10 GRIB EDITOR (PICTURE2) ................................................ 65
  11 FILER ................................................................................... 66
  UTILITIES .................................................................................. 67

Meta Kernel

Contents
2.14.1 Assigning 0 to a register ................................................................................................................ 38
2.14.2 Loading a value in A or C .................................................................................................................. 38
2.14.3 Loading a register value into another register ................................................................................. 39
2.14.4 Exchange between two registers ..................................................................................................... 39
2.14.5 Addition ............................................................................................................................................ 39
2.14.6 Subtraction ...................................................................................................................................... 39
2.14.7 Increment and decrement .................................................................................................................. 39
2.14.8 Right nibbles shifting (divide by 16) ................................................................................................. 39
2.14.9 Left nibbles shifting (multiply by 16) ............................................................................................... 39
2.14.10 Right bit shifting (divide by 2) ........................................................................................................ 39
2.14.11 Right circular nibble shifting ......................................................................................................... 39
2.14.12 Left circular nibble shifting ........................................................................................................... 39
2.14.13 Logical AND .................................................................................................................................. 39
2.14.14 Logical OR ................................................................................................................................... 39
2.14.15 Logical NOT ................................................................................................................................. 39
2.14.16 Mathematical NOT ....................................................................................................................... 40
2.14.17 Loading value into a R Register ....................................................................................................... 40
2.14.18 Loading value into A or C from a R register .................................................................................. 40
2.14.19 Exchange between A or C and a R register ................................................................................... 40
2.14.20 Memory write (POKE) .................................................................................................................. 40
2.14.21 Memory read (PEEK) .................................................................................................................... 40
2.14.22 D0 and D1 modifications ................................................................................................................ 40
   a) Loading D0 and D1 ............................................................................................................................. 40
   b) Exchanges between A or C and D0 or D1 .......................................................................................... 40
      • Loading A or C, field A, into D0 or D1 .......................................................................................... 40
      • Loading the four low nibbles of A or C into D0 or D1 ................................................................. 40
      • Exchanging A or C, field A, and D0 or D1 .................................................................................. 40
      • Exchanging the 4 first nibbles of A or C and D0 or D1 ............................................................ 40
   c) Increment and decrement of D0 and D1 ........................................................................................... 40
2.14.23 Working registers tests .................................................................................................................... 41
   a) Equality and inequality tests ............................................................................................................. 41
   b) Lower and greater tests ..................................................................................................................... 41
   c) Nullity tests ...................................................................................................................................... 41
2.14.24 Working with some bits of A or C register ...................................................................................... 41
2.14.25 Operations on PC ......................................................................................................................... 41
2.14.26 Working with the Hardware Status Register ................................................................................ 41
2.14.27 Working with P ............................................................................................................................... 41
2.14.28 Jump instructions ........................................................................................................................... 41
2.14.29 Exchanges between C and RSTK ................................................................................................. 42
2.14.30 Input / output instructions .......................................................................................................... 42
2.14.31 Processor control instructions ..................................................................................................... 42
2.14.32 New instructions of Masd ............................................................................................................. 42
2.15 Masd directives .................................................................................................................................. 42
2.16 Error messages ................................................................................................................................... 43
3 DISASSEMBLER ..................................................................................................................................... 43
4 MISCELLANEOUS UTILITIES ............................................................................................................ 43
4.1 \[H\] PEER ............................................................................................................................................. 44
4.2 \[H\] POKE ........................................................................................................................................... 44
4.3 \[S\] SREV .......................................................................................................................................... 44
4.4 \[C\] CD ................................................................................................................................................ 44
4.5 \[D\] \[CD\] ........................................................................................................................................... 44
4.6 \[A\] \[CD\] ............................................................................................................................................ 44
4.7 \[A\] \[A\] .............................................................................................................................................. 44
4.8 \[A\] POKE ........................................................................................................................................... 44
4.9 PEEK ................................................................................................................................................... 44
4.10 POKE ................................................................................................................................................. 44
4.11 SREV ................................................................................................................................................. 44
4.12 PRG \[LST\] PRG \[LST\] .......................................................................................................................... 44
4.13 \[RAM\] ............................................................................................................................................. 44
4.14 APEEK .............................................................................................................................................. 44
4.15 \[SB\] \[SB\] ......................................................................................................................................... 44
4.16 \[GROB2\] \[GROB2\] .............................................................................................................................. 44
4.17 \[HEADER\] \[HEADER\] ....................................................................................................................... 45
4.18 HEADER \[HEADER\] .......................................................................................................................... 45
4.19 INPUT2 \[INPUT\] ................................................................................................................................ 45
4.20 Disp2 \[Disp2\] ................................................................................................................................... 45
APPENDIXES................................................................................................................................. 48

1 INTRODUCTION TO ASSEMBLY LANGUAGE ................................................................................. 48
2 INTRODUCTION TO SYSTEM RPL ...................................................................................................... 48
3 TIPS AND TRICKS............................................................................................................................ 48
  3.1 Meta Kernel and UFL .................................................................................................................. 48
  3.2 STARTED and EXITED examples.............................................................................................. 48
    3.2.1 Remove the header display ................................................................................................. 48
    3.2.2 Edit compressed files ......................................................................................................... 48
  3.3 Meta Kernel and Erable ............................................................................................................. 49
    3.3.1 STARTOFF and TOFF example ......................................................................................... 50
4 THE MK FOR PROGRAMMERS ............................................................................................................ 50
  4.1 Customizing the Filer ................................................................................................................. 50
    4.1.1 Overview .......................................................................................................................... 50
    4.1.2 FILER CUSTOM Format .................................................................................................... 50
    4.1.3 Explanation of each component ......................................................................................... 50
      a) System Binary 1 and GROB ............................................................................................... 50
      b) System binary 2 .................................................................................................................. 50
      c) System binary 3 .................................................................................................................. 50
      d) Internal calls ..................................................................................................................... 51
      e) Custom program ............................................................................................................... 51
      f) Explanation for the string of addresses ............................................................................ 52
      g) Explanations about the name ............................................................................................ 52
      h) Warning and very important points .................................................................................. 52
  4.2 Machine Language entry points list .......................................................................................... 52
    4.2.1 MINI_DISP ....................................................................................................................... 52
    4.2.2 MINI_FONT_ADDRESS .................................................................................................... 52
    4.2.3 DISPLAY_SBR .................................................................................................................. 52
    4.2.4 EDIT_SBR .......................................................................................................................... 52
    4.2.5 FILER_ADR ....................................................................................................................... 52
    4.2.6 SCAN_FONT ......................................................................................................................... 53
    4.2.7 RECONFIG .......................................................................................................................... 53
    4.2.8 DECONFIG .......................................................................................................................... 53
    4.2.9 CMD_SIZE ......................................................................................................................... 53
    4.2.10 GET_PATH ......................................................................................................................... 53
    4.2.11 DECONFIG_RAM .............................................................................................................. 53
    4.2.12 RECONFIG_RAM .............................................................................................................. 53
    4.2.13 GET_FONT .......................................................................................................................... 53
4.3 SysRPL entry points......................................................................................................... 57
4.3.24 LOAD.ARG ................................................................................................................. 60
4.3.23 SAVE.ARG............................................................................................................... 60
4.3.22 MATRIX.ADR ........................................................................................................... 60
4.3.21 COMMANDLINE.ADR............................................................................................. 60
4.3.19 UPSTACK.ADR........................................................................................................... 59
4.3.16 KER_PARAM ............................................................................................................ 59
4.3.14 DISP_CALL ............................................................................................................. 59
4.3.13 CMP_PLUS ............................................................................................................. 59
4.3.9 SURPRISE ................................................................................................................ 58
4.3.7 FILER_FRCL............................................................................................................. 58
4.3.5 DIMS....................................................................................................................... 58
4.2.14 INIT_DISPLAY_LINE .............................................................................................. 53
4.2.15 CHANGE_FLAG ....................................................................................................... 53
4.2.16 GET_ASCII_KEY ...................................................................................................... 54
4.2.17 GET_KEY ................................................................................................................ 54
4.2.18 RUN_KEY ............................................................................................................... 54
4.2.19 MINI_DISP_VAL .................................................................................................... 54
4.2.20 MINI_DISP_AWP ................................................................................................... 54
4.2.21 TRUP ..................................................................................................................... 54
4.2.22 TRDN .................................................................................................................... 54
4.2.23 ZEROM .................................................................................................................. 54
4.2.24 SCAN_KEY ............................................................................................................ 54
4.2.25 NEW_ADR ............................................................................................................ 54
4.2.26 RESIZE_PLUS ...................................................................................................... 54
4.2.27 C=IN3 .................................................................................................................. 55
4.2.28 A=IN3 .................................................................................................................. 55
4.2.29 OUT=C=IN3 ........................................................................................................ 55
4.2.30 OUT=CA=IN3 ....................................................................................................... 55
4.2.31 DISP_DEC ............................................................................................................. 55
4.2.32 MULT_BAC .......................................................................................................... 55
4.2.33 GREY? .................................................................................................................. 55
4.2.34 BEEP .................................................................................................................... 55
4.2.35 KEY_REPEAT ........................................................................................................ 55
4.2.36 KEY_NO .............................................................................................................. 55
4.2.37 HEX_DEC ............................................................................................................. 55
4.2.38 A_DIV_C .............................................................................................................. 55
4.2.39 SET_BIT .............................................................................................................. 55
4.2.40 CLEAR_BIT ........................................................................................................ 56
4.2.41 BIT? ..................................................................................................................... 56
4.2.42 BZU.SBR ............................................................................................................. 56
4.2.43 INV.ZONE .......................................................................................................... 56
4.2.44 OFF.SBR ............................................................................................................. 56
4.2.45 InitTable .............................................................................................................. 56
4.2.46 GetAdr ................................................................................................................ 56
4.2.47 GetText .............................................................................................................. 56
4.2.48 GetFirst .............................................................................................................. 57
4.2.49 GetNext ............................................................................................................. 57
4.2.50 GetFirstAdr ....................................................................................................... 57
4.2.51 GetNextAdr ....................................................................................................... 57
4.2.52 RecPath ............................................................................................................. 57
4.3 SysRPL entry points...................................................................................................... 57
4.3.1 GET_INDEX ............................................................................................................ 57
4.3.2 GET.Y.I ................................................................................................................ 57
4.3.3 GET.X.Y.I .......................................................................................................... 58
4.3.4 PUT.X.Y .............................................................................................................. 58
4.3.5 DIMS .................................................................................................................... 58
4.3.6 FILER_FNAME ..................................................................................................... 58
4.3.7 FILER_FRCL ......................................................................................................... 58
4.3.8 FILER_NEXTADR ................................................................................................. 58
4.3.9 SURPRISE .......................................................................................................... 58
4.3.10 INPUT ................................................................................................................ 58
4.3.11 CTRL_LOOP ....................................................................................................... 59
4.3.12 INIT_CMD ........................................................................................................... 59
4.3.13 CMP_PLUS ....................................................................................................... 59
4.3.14 DISP_CALL ....................................................................................................... 59
4.3.15 BZU ................................................................................................................... 59
4.3.16 KER_PARAM ...................................................................................................... 59
4.3.17 MINI_EDITOR ................................................................................................. 59
4.3.18 KEY_PARAM ...................................................................................................... 59
4.3.19 UPSTACK.ADR ................................................................................................. 59
4.3.20 WAITKEY.ADR ................................................................................................. 59
4.3.21 COMMANDLINE.ADR ..................................................................................... 60
4.3.22 MATRIX.ADR ................................................................................................. 60
4.3.23 SAVE.ARG ....................................................................................................... 60
4.3.24 LOAD.ARG ....................................................................................................... 60
5 MetaKernel 3.0 : PREVIEW ......................................................................................... 61
Using the Meta Kernel

1 Installation

Switch off your calculator by pressing \( \text{OFF} \).
Put the ROM card into the slot 1 of the HP48 (it must be the port 1).
Turn the HP48 on.
After a short time, the Meta Kernel is installed:

2 General presentation

The Meta Kernel is an application card that changes the standard environment. A good knowledge of the standard environment is required, as the MK uses it, and adds several functions. Please read the HP48 GX User’s Guide about it.

With the new stack display and command line, new utilities are provided:

- Equation editor (EQW) ...............................................Page 17
- Matrix editor (MATR) ..................................................Page 23
- Grob editor (PICTURE2) ............................................Page 24
- File utility (FILER).......................................................Page 27
- Symbolic assembler (MASD)......................................Page 32
- Library maker .............................................................Page 35
- System RPL Development Kit ............................ Pages 16, 36

3 Stack display

3.1 Display

The display is divided in three parts: (the command line belongs to the stack)

\[ \text{Status area} \quad \rightarrow \quad \text{Stack} \quad \rightarrow \quad \text{Menu} \]

The stack and status area can be resized in height, see below.

3.1.1 Status area

The status area shows the current HP48 GX state.
Error messages, current path, calculator modes are displayed in this area.

Here are the new state flags:

- \( \text{DEG} \) : Degrees
- \( \text{RAD} \) : Radians
- \( \text{GRD} \) : Grads
- \( \text{REC} \) : Rectangular coordinates
- \( \text{POL} \) : Polar coordinates
- \( \text{SHP} \) : Spherical coordinates
- \( \text{ALG} \) : Algebraic entry mode
- \( \text{PRG} \) : Program entry mode
- \( \text{UER} \) : User keyboard
- \( \text{USR} \) : User keyboard for the next keystroke
- \( \text{MEN} \) : Hexadecimal base
- \( \text{DEC} \) : Decimal base
- \( \text{OCT} \) : Octal base
- \( \text{BIN} \) : Binary base
- \( \text{ROM} \) : ROM or write-protected
- \( \text{RAM} \) : RAM in port 2
- \( \text{WRI} \) : Read-write RAM card in port 2

The status area occupies two text lines by default. Its height can be changed to one or zero lines with the MK command \( \text{MK} \), consult Page 45.

3.1.2 Stack

With the MK, up to nine lines can be viewed on the stack, as you can change the font height (6, 7 or 8 pixels) and the status area height.

The maximum number of lines displayed by objects can be changed with \( \text{\#H\#ISP} \), consult Page 46.

More configuration capabilities are provided by the Meta Kernel system flags (MKSF), consult Page 11.

Algebraics and grobs may be displayed directly in the stack, use the MKSF to configure how and when they are displayed.
3.1.3 Menu labels

Menu-label style depends on the type of the stored object:

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directory</td>
<td>Directory</td>
</tr>
<tr>
<td>[Prg] Grob</td>
<td>Program, code</td>
</tr>
<tr>
<td>[Rml] Real</td>
<td>Real, complex, algebraic expression, matrix</td>
</tr>
<tr>
<td>[Lst] List</td>
<td>List, variable, binary integer</td>
</tr>
<tr>
<td>[Charr] String</td>
<td>Character string</td>
</tr>
</tbody>
</table>

By default, labels are written using the **MK** mini font. If the **MK** system flag -20 is set (with \-20 SF2), standard menu font is used instead:

3.2 Display configuration

New display fonts can be created and used. The number of lines displayed on the stack differs with the font height.

With a font8 (8 pixels height), 5 to 7 lines may be viewed; with a font7, 6 to 8 lines; with a font6, 7 to 9 lines.

3.2.1 Changing the display font

To use a different font, put it on the stack, and execute \-FON\ T consult Page 46.

Fonts are displayed on the stack like this:

```
8 1 SYSTEM 8
```

Example: \-FON\ T SYSTEM 8, this is the standard display font for the **MK**.

The first number (8) is the font height in pixels, the second number (8) is the id of the font, then the name (SYSTEM 8).

Two examples with a font8 and a font6:

```
8 1 SYSTEM 8
6 1 SYSTEM 6
```

If the current font is a font6 (6 pixel-high), the **MK** system flags -8 and -9 force respectively the stack display and the command line to use the mini font:

3.2.2 Modifying the current font

The current display font may be modified with the CHARS menu (\-CHARS\). Consult Page 31.

On the floppy disk, there are two programs, named GRB2FNT.PRG and FNT2GRB.PRG, which convert **MK** fonts to or from **common** 8x2048 grobs. Consult Page 46.

3.2.3 Changing the mini font

It is possible to change the **MK** mini font, used in the status area and the menu labels.

Use the commands **MINIFONT** \- and \+MINIFONT. A mini font is a **Library** Data object. This format is very popular in the HP48's world, particularly with programs which use UFL (Universal Font Library). In order to know how to use the UFL with the **Meta Kernel** check the column Tricks and Tips.

4 Configuration
4.1 MK variables

Some programs can be executed on some events, in place of the default programs. They must be stored in the HOME directory.

<table>
<thead>
<tr>
<th>STARTUP</th>
<th>Executed at each reboot (ON - OFF...).</th>
</tr>
</thead>
<tbody>
<tr>
<td>STARTED</td>
<td>Executed just before entering in command line for editing an object. The object to be edited can be found on stack level 1. This program must leave the object to be edited on the stack level 1.</td>
</tr>
<tr>
<td>EXITED</td>
<td>Executed when the user stops the command line editing. Level 2 contains the object and level 1 TRUE if the user has pressed ENTER or only FALSE for ON. For example, these two programs can be used to change the display while editing (try STARTED: « &amp; 0 +HEADER » and EXITED: « 2 +HEADER »).</td>
</tr>
<tr>
<td>STARTOFF</td>
<td>Executed when the automatic OFF is called, after a certain amount of idle time (You can make a screen saver! But remember, the HP48 GX works on batteries...).</td>
</tr>
<tr>
<td>TOFF</td>
<td>Defines the amount of idle time before an automatic OFF (or STARTOFF). It must be a binary integer, in Ticks (1 s = 8192 Ticks).</td>
</tr>
<tr>
<td>STARTEXT</td>
<td>Contains a string that is the path (using the MASD syntax) to the entries points list. Example: &quot;4/DIR1/TABLE.EXT&quot;, &quot;8&quot; will refer to the table TABLE.EXT in Port 4, in the backup DIR1. For more details, please read page 36.</td>
</tr>
<tr>
<td>STARTSEND</td>
<td>Used by the filer for file transfer. By default, Kermit is used. If STARTSEND exists, it is called with the object on level 2 and the name on level 1. Consult page 27.</td>
</tr>
<tr>
<td>STARTERR</td>
<td>Used for the error messages display. The error is found in two character strings on levels 2 and 1.</td>
</tr>
<tr>
<td>STARTEOV</td>
<td>Called by EQW (equation editor) when CST is pressed.</td>
</tr>
<tr>
<td>FILER.CUSTOM</td>
<td>Called by the Filer when CST is pressed.</td>
</tr>
</tbody>
</table>

4.2 MK user and system flags (MKUF, MKSF)

The Meta Kernel adds a new series of flags.

Use CF2, SF2, FC?2, FS?2, FC?C2, FS?C2 like the functions CF, SF...

By default, all flags are cleared.

MK System flags -1 to -64 are used by the MK to change parameters.

MK User flags 1 to 64 can be used by any program that runs under the MK.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Cleared (CF2)</th>
<th>Set (SF2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Multiline for stack display at all levels.</td>
<td>No multiline for stack display (except level 1).</td>
</tr>
<tr>
<td>-2</td>
<td>Strings displayed on multiple lines.</td>
<td>Strings displayed on only one line.</td>
</tr>
<tr>
<td>-3</td>
<td>Digital clock in status area.</td>
<td>Analog clock in status area.</td>
</tr>
<tr>
<td>-4</td>
<td>No auto-indentation in command line editing.</td>
<td>Auto-indentation in command line editing.</td>
</tr>
<tr>
<td>-5</td>
<td>Cursor can not go out of the text.</td>
<td>Full page editing.</td>
</tr>
<tr>
<td>-6</td>
<td>Strings take only one line for +GR0B.</td>
<td>Multi-line strings for +GR0B.</td>
</tr>
<tr>
<td>-7</td>
<td>RSH+ with addresses.</td>
<td>RSH+ without the addresses (makes labels).</td>
</tr>
<tr>
<td>-8</td>
<td>The current font is normally used in stack display.</td>
<td>Mini-font in stack display (the current font must be a font6).</td>
</tr>
<tr>
<td>-9</td>
<td>The current font is normally used in command line editing.</td>
<td>Mini-font in command line editing (the current font must be a font6).</td>
</tr>
<tr>
<td>-10</td>
<td>Rightmost stack display format.</td>
<td>Leftmost stack display format.</td>
</tr>
<tr>
<td>Line</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>-11</td>
<td>Silent keystrokes. Keystrokes click.</td>
<td></td>
</tr>
<tr>
<td>-12</td>
<td>No purge confirmation in the Filer. Purge confirmation in the Filer.</td>
<td></td>
</tr>
<tr>
<td>-13</td>
<td>(Reserved for internal use).</td>
<td></td>
</tr>
<tr>
<td>-14</td>
<td>(Reserved for internal use).</td>
<td></td>
</tr>
<tr>
<td>-16</td>
<td>Current font for EOQ stack display. Mini-font for EOQ stack display.</td>
<td></td>
</tr>
<tr>
<td>-18</td>
<td>Current font for algebraics editing in E0Q. Mini-font for algebraics editing in E0Q.</td>
<td></td>
</tr>
<tr>
<td>-19</td>
<td>On-stack grobs display. Standard stack display for grobs.</td>
<td></td>
</tr>
<tr>
<td>-20</td>
<td>MK Mini-font menu labels. Standard menu labels.</td>
<td></td>
</tr>
<tr>
<td>-21</td>
<td>Normal stack display. Display the stack using the HP SysRPL syntax.</td>
<td></td>
</tr>
<tr>
<td>-22</td>
<td>$S2$ adds &quot;INO CODE</td>
<td>CR</td>
</tr>
<tr>
<td>-23</td>
<td>The stack display is not recursive.</td>
<td></td>
</tr>
<tr>
<td>-24</td>
<td>Object display in the command line is not recursive. (See flag –23)</td>
<td></td>
</tr>
<tr>
<td>-25</td>
<td>In the normal stack display (-21 CF2), displays the addresses of all the unknown objects:</td>
<td></td>
</tr>
<tr>
<td>-26</td>
<td>The CHOOSE2 boxes use the current font. The CHOOSE2 boxes use the mini-font.</td>
<td></td>
</tr>
</tbody>
</table>
5 Interactive stack

The interactive stack works the same way as the standard one, but at a greater speed, and it is possible to edit many objects simultaneously: in the command line editor, it is possible to go temporarily to the interactive stack, and to edit another object recursively. It is also possible to HRLT the editing environment.

Interactive Stack Operations: (n indicates the current level)

- ECHO: Copies the current level object on the command line, at the cursor position.
- INFO: Shows information about the current level object (size, CRC...).
- INFO: Edits the current level object with the best environment (EDITB).
- INFO: Edits the object whose name is on the current level, with the best environment (VISITB).
- PICK: Equivalent to nPICK.
- ROLL: Equivalent to nROLL.
- ROLLD: Equivalent to nROLD.
- LIST: Equivalent to nLIST.
- DUPO: Equivalent to nDUPN.
- DROF: Equivalent to nDROPN.
- LEVEL: Puts the number of the current level on level 1.
- GOTO: Goes directly to a given level.
- KEEP: Clears all the levels upper than the current level (Therefore it KEEPs the first stack levels up to the current one).
- SLOW / FAST: Selects the cursor speed.

6 Command line

6.1 Generalities

The Meta Kernel improves editing with new functions: COPY/PASTE, FIND/REPLACE, text style (Bold, italic, underline, inverse). Styles can only be used inside strings (they will be ignored and lost in any other object).

You can edit an object by pressing when the object is on stack level 1. There are other ways to edit an object (explained after).

6.2 Editing menu

There are four pages in the menu, plus the Style menu.

6.2.1 Major options

- GOTO: Goes to a given line number.
- GOTO: Goes to a given position (in number of characters from the beginning).
- GOTO: Displays the menu of the text, which is composed of every line that begins with the character *, useful in assembly language.
- BEG: Goes to the beginning of the selection.
- END: Goes to the end of the selection.
- DEL L: Clears the current line.
- SLOW / FAST: Selects the cursor speed.
- INFO: Displays information about the current text.
- INFO: Edits the selection with the best environment (EDITB).
6.2.2 Operations on the selection

Note: the selection marks are lost when the text is modified.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BEGIN</code></td>
<td>Sets the beginning of the selection.</td>
</tr>
<tr>
<td><code>END</code></td>
<td>Sets the end of the selection.</td>
</tr>
<tr>
<td><code>COPY</code></td>
<td>Copies the selection on the stack level 1.</td>
</tr>
<tr>
<td><code>PASTE</code></td>
<td>Copies the selection on the stack level 1.</td>
</tr>
<tr>
<td><code>PASTE</code></td>
<td>Puts the object on the stack level 1 into the text at the cursor position.</td>
</tr>
<tr>
<td><code>DEL</code></td>
<td>Puts the object on the stack level 1 into the text at the cursor position, and deletes it from the stack.</td>
</tr>
<tr>
<td><code>DEL</code></td>
<td>Clears the selection.</td>
</tr>
</tbody>
</table>

6.2.3 Search operations

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FIND</code></td>
<td>Displays a screen where you can type the search string. When you press enter, this string is put on stack level 1. The character <code>/</code> stands for any character. Then the selection is placed on the first occurrence of this string after the cursor position.</td>
</tr>
<tr>
<td><code>REPL</code></td>
<td>Displays a screen where you can type the search string and the replace string. The search string is put on stack level 1, and the replace string on level 2. Then the selection is placed on the first occurrence of the search string. The change is not done yet!</td>
</tr>
<tr>
<td><code>NEXT</code></td>
<td>Selects the next occurrence of the search string (the string on stack level 1).</td>
</tr>
<tr>
<td><code>R</code></td>
<td>Replaces the selection by the replace string (the string on stack level 2).</td>
</tr>
<tr>
<td><code>R:NEXT</code></td>
<td>Equivalent to <code>R</code>, then <code>NEXT</code>. So it makes one change, and goes to the next occurrence of the search string.</td>
</tr>
<tr>
<td><code>ALL</code></td>
<td>Equivalent to <code>R</code>, and <code>NEXT</code>, repeated until the end of the text. So every occurrence of the search string is replaced by the replace string, from the cursor position to the end of the text, without confirmation.</td>
</tr>
</tbody>
</table>

6.2.4 Miscellaneous

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SINTE</code></td>
<td>Goes to the beginning of the previous word.</td>
</tr>
<tr>
<td><code>SINT</code></td>
<td>Goes to the beginning of the next word.</td>
</tr>
<tr>
<td><code>SDEL</code></td>
<td>Clears the text between the cursor and the beginning of the previous word.</td>
</tr>
<tr>
<td><code>DEL</code></td>
<td>Clears the text between the cursor and the beginning of the next word.</td>
</tr>
<tr>
<td><code>STY</code></td>
<td>Displays the style menu.</td>
</tr>
<tr>
<td><code>STH</code></td>
<td>Goes to the interactive stack. Consult Page 13.</td>
</tr>
<tr>
<td><code>STUI</code></td>
<td>Displays the selection and replaces it with the result. E.g. if <code>20 SIN</code> is selected, it will be replaced by <code>342020143326</code>.</td>
</tr>
<tr>
<td><code>STUR</code></td>
<td>Equivalent to <code>HALT</code>. The current editing is suspended, you can do what you want (except a reboot) and get back to the text with <code>CONTINUE</code>.</td>
</tr>
</tbody>
</table>

6.3 Styles

6.3.1 Generalities

Styles can be used inside text strings. There are four styles: Bold, italic, underline, inverse.

Several styles can be used on the same portion of text.

More than one font can also be used in the same text (but they must have the same height).

6.3.2 Style menu

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BOL</code></td>
<td>Sets/clears the bold style.</td>
</tr>
<tr>
<td><code>ITAL</code></td>
<td>Sets/clears the italic style.</td>
</tr>
<tr>
<td><code>UNDE</code></td>
<td>Sets/clears the underlined style.</td>
</tr>
<tr>
<td><code>REV</code></td>
<td>Sets/clears the reversed style.</td>
</tr>
<tr>
<td><code>FON</code></td>
<td>Displays the font list.</td>
</tr>
<tr>
<td><code>EDIT</code></td>
<td>Gets back to the Edit menu.</td>
</tr>
</tbody>
</table>

6.3.3 Using styles

You can modify the style of the current selected text by pressing a style menu key. If there is no selection, the text you will then type will be of the chosen style (If you move the cursor, the style will be the one under the cursor).

6.3.4 Using fonts

If you press the `FON` menu key, a choose box shows all the fonts which have the same height as the current editing font.

If there is a selection, the font will be applied on this selection, otherwise it will be used for the text you will then type (If you move the cursor, the font will be the one under the cursor).
The Meta Kernel can use up to 247 fonts inside one text.

For a font to be recognized by the MK (so that it will appear in the FONT choose box), it just has to be saved in the HOME directory, or in a port (0, 2 to 31)

If several fonts use the same ID, the MK only takes the first one, in the order of the search (HOME, then 0, 2 ... 31). ID 0 is reserved for the system font. ID > 247 will be ignored.

Example:

Type the text: THIS IS A SIMPLE EXAMPLE

Put the cursor on the 11th character (S in SIMPLE). Press the menu key

Press six times. Press

Go to the menu and press

Now, select the word EXAMPLE

Go to the menu and press (There may be other fonts than shown below)

Choose the font you want (here ROBOT 8) with the up and down arrows and press

6.4 Miscellaneous command line options

6.4.1 Full screen mode

By default, if you press the key at the end of a line, the cursor will be placed on the first character of the next line. With the MK, it is possible to go 'out' of the lines, with the MK system flag -5.

Type -5 SF2 to enable the full screen mode (if you like this mode, a good idea is to place a -5 SF2 in the program STARTUP). If you go beyond the end of a line and press a character key, spaces will automatically be inserted before the character you type.

6.4.2 Auto-indent mode

By default, when you press to insert a carriage return, the cursor is then always placed on the first column of the display. The MK implements the auto-indent mode, so that when you type a carriage return, the cursor will be placed under the first character of the upper line (Useful when programming).

Type -4 SF2 to enable auto-indent mode.
6.4.3 Special keystrokes for System RPL

In the command line, some keystrokes help to type characters that are specific to System RPL, and also to easily find System RPL commands only by typing the first letters of the command.

These keystrokes are detailed on page 37.

6.4.4 Big strings editing

As specified in the HP48 GX User’s Manual, you can not type character strings that are bigger than the half of the free memory.

The MK uses a new memory handler, so that when you edit strings (and only strings), you can have more characters than the half of the free memory, and still press \( \text{ENTER} \), because the MK does not evaluate the string, but places it directly on the stack. You will never get an Insufficient Memory when validating a character string.

For other objects types, the MK built-in \(+\text{STR}\) will usually works if there is more or less as much free memory as the object size, so the object may be viewed. But when \( \text{ENTER} \) is pressed, the \( \text{STR} \) may have Insufficient Memory.

6.5 Shortcuts

The Shortcuts are available by pressing simultaneously a SHIFT key and another one.

- **Mode**
  - Insert the character \( \backslash \)
- **MEM**
  - Suspend the current environment
- **EVAL**
  - Page Up
  - Evaluate the current line and replace it with the result. Example:
    - Begin
  - Page Left
  - Page Down
  - Page Right
  - **4**
    - BEGIN
  - **5**
    - END
  - **6**
    - Select the current line
  - **1**
    - Extend/Reduce the selection to the left
  - **3**
    - Extend/Reduce the selection to the right
  - **a**
    - Insert and indent a SystemRPL program bloc (see the MASD chapter for more information)
  - **a**
    - Auto-Completion (see the MASD chapter for more information)

7 Command library

A command library is a special library which extends the \( \text{STR} \) function. The MK command library adds new keywords (they can be typed on the command line and used in programs):

- **TRUE**
  - External Boolean true.
- **FALSE**
  - External Boolean false.
- **$**
  - External of the given address. E.g. \# \text{3188h} (external DUP). To type \#, press \( \text{a} \) \( \text{c} \) \( \text{4} \).
- **e**
  - External of the given address. \# \text{3188} is equivalent to \# \text{3188h}. To type \( e \), press \( \text{a} \) \( \text{c} \) \( \text{e} \).
- **b**
  - System binary of the given number. To type \text{b}, press \( \text{a} \) \( \text{c} \) \( \text{6} \).
- **K**
  - Character of the given ASCII code.
- **XLIB**
  - Program \#\text{PrgNumber} of the library \#\text{LibNumber}.
- **INCLUDE**
  - Includes a given variable content in the line.
- **PRG**
  - External program beginning (\#: \text{02D90h}), a \text{END} must close the program.
- **-**
  - NoEval (\#: \text{06E97h}), which causes the next object not to be evaluated. To type \( - \), press \( \text{a} \) \( \text{c} \) \( \text{y} \).
- **PROG**
  - To edit an external program, begin it with \text{PROG}, so it won’t be evaluated when \( \text{ENTER} \) is pressed.
8 Equation editor

The program EQW can easily create, edit and manipulate algebraic objects, in their hand-written style.

For example, here is a mathematical equation:

\[ E = \int_{0}^{\frac{1}{X}} |F(t)| dt \]

Here is how it usually appears on the stack:

\[ E = \int_{0}^{\frac{1}{X}} |F(t)| dt \]

And now, here is the same equation typed with EQW:

This EQW display is also used in the stack display:

8.1 First view of EQW

In this document, ‘equation’ stands for all algebraic objects.

8.1.1 Introduction

EQW is a special environment in which the keyboard is redefined and limited to specific operations.

8.1.2 Modes

EQW provides three modes of operation:

• Selection mode

\[ E = \int_{0}^{\frac{1}{X}} |F(t)| dt \]

\[ E = \int_{0}^{\frac{1}{X}} |F(t)| dt \]

• Editing mode

\[ E = \int_{\text{ABC}}^{\frac{1}{X}} |F(t)| dt \]

In most cases, selection and edit modes are used to enter equations.

• Cursor mode, used to navigate rapidly through the equation.

8.1.3 Editing

During editing, keys behave in different ways, depending on the current mode.

In selection mode, a part of the equation is inverted or boxed. If a function key is pushed, the function is applied on the selection.
In edit mode, a flashing cursor is ready to insert new characters.

The next sections explain how to enter and edit an equation.

8.2 Creating equations

8.2.1 Running EQW

Type ` ` to run EQW with a new equation.

8.2.2 Exiting EQW

To validate the current equation on the stack, press ` `. To get out of EQW, and lose all changes made on the equation, press ` `.

There may be some times when EQW can’t draw the equation as fast as you type, but all keystrokes are processed, none are lost.

In this case, EQW draws the equation only when there is enough time to do so.

8.2.3 Numbers and names

Type real numbers and names the same way as in the command line.

In edit mode (with the flashing cursor), the character `/` is directly available by pressing ` `. Note: in EQW, the ` ` key locks the alpha mode by pressing it once (corresponding to ` `). Therefore, to type a name, either hold ` ` and press the character keys, or press ` ` once, then press the character keys and press ` ` to unlock the alpha mode.

8.2.4 Addition, subtraction, and multiplication

To enter `/` and `/`, press ` ` and ` `. The operation is applied on the edited number or name, or the current selection.

To enter implicit multiply, do not press ` `. An implicit multiply is automatically inserted in the following cases:

- A number followed by an alpha character or a prefix function (a function whose name is before arguments, for example ` `).
- An alpha character followed by a prefix function.

8.2.5 Complex numbers and expressions

To enter a complex number or expression, type in the real part, then ` ` and then the imaginary part.

A complex expression is considered as a two-argument function (equivalent to `/`).

A complex number is stored directly in the final equation, whereas a complex expression like ` ` is converted to ` `.  

8.2.6 Moving in the equation

Before going further, here are the movement keys. In fact, entering an equation with EQW is not far from the RPN way: no parenthesis are needed. For example, to type `(x+5)*2`, press ` `, ` `, ` ` and ` `. Then select ` ` by typing ` `, and then type ` `.

See the examples at the end of this chapter to handle more cases.

Arrow keys move the selection. They behave different ways, depending on the type of selection.

Here are the movement keys when the selection is inverted:

- ` ` selects the first argument of the current function, or if the selection is already a number or a name, then selection becomes a box.
- ` ` selects the upper function of the selection.
- ` ` and ` ` select the previous or next argument of the upper function.

Here are the movement keys when the selection is in a box:

- ` ` goes back into inverted selection mode.
- ` ` and ` ` select the previous or next number or name, in hierarchical order. This keys are useful to rapidly move around the whole equation.

To rapidly select a part of the equation, even far from the selection, use the cursor mode (see the Cursor Mode section).

8.2.7 Division

Press ` ` to insert a division.

Division is applied on the edited object or the current selection.

To ‘go out’ of the division, use ` ` or ` ` until the whole division is selected.

8.2.8 Exponents

` ` begins the exponent on the edited object or the selection.

Use ` ` or ` ` to go out of the exponent.

8.2.9 Square root

- In selection mode, ` ` applies a square root on the selection.
- In edit mode, ` ` inserts a square root, eventually with an implied multiply.
8.2.10 Nth square root

- In selection mode, \(\sqrt[n]{x}\) places the selection under the root. Press \(\downarrow\) and type the outside term. To exchange the two arguments, read the Modifications section.
- In edit mode, \(\sqrt[n]{x}\) inserts a root. Type the outside term. Press \(\uparrow\) and then type the inside expression.

8.2.11 Parenthesized-argument functions

- In selection mode, press the function key to apply it on the selection. If the function needs more than one argument, the selection is used as the first one. press \(\uparrow\) to go and type the next arguments.
- In edit mode, press the function key or type its name followed by \(\downarrow\), and then the arguments.

8.2.12 User functions

These functions are created by the user and are not handled directly by the HP 48GX.

Type the function name, press \(\downarrow\) and type the first argument.
To add an argument, press \(\uparrow\).

8.2.13 Parenthesized terms

EQW draws equations with the fewest possible parentheses. To add explicit parentheses around the selection, press \(\downarrow\).

8.2.14 Differentiation

- In selection mode, press \(\text{\texttt{\textbf{\textbackslash SIN}}}\) to insert a differentiate sign, the selection is the function to be differentiated. Press \(\downarrow\) and type the variable of differentiation.
- In edit mode, press \(\text{\texttt{\textbf{\textbackslash SIN}}}\) to insert a differentiate sign, type the variable of differentiation, press \(\uparrow\) and type the equation to be differentiated.

See the Evaluations section to discover how to differentiate a function without exiting from EQW.

8.2.15 Integration

- In selection mode, press \(\text{\texttt{\textbf{\textbackslash COS}}}\) to add an integral sign. The current selection is the equation to be integrated. Press \(\downarrow\) and \(\uparrow\) to move around the integral arguments and type them.
- In edit mode, press \(\text{\texttt{\textbf{\textbackslash COS}}}\) to insert an integral sign. Type the arguments, press \(\uparrow\) to move to the next one.

8.2.16 Summations

- In selection mode, press \(\text{\texttt{\textbf{\textbackslash TAN}}}\) to add a summation sign. The selection is the function to be summed. Press \(\downarrow\) and \(\uparrow\) to move around the arguments and type them.
- In edit mode, press \(\text{\texttt{\textbf{\textbackslash TAN}}}\) to insert a summation sign. Type the arguments, press \(\uparrow\) to move to the next one.

8.2.17 Where function \(\mid\)

- In selection mode, press \(\text{\texttt{\textbf{\textbackslash VAR}}}\) to add a where function. The current selection is the base expression. Press \(\uparrow\) to move to the next argument (the variable and its value).
- In edit mode, press \(\text{\texttt{\textbf{\textbackslash VAR}}}\) to insert a where sign. Type the arguments, press \(\uparrow\) to move to the next one.

8.3 Manipulating equations

8.3.1 Modifying equations with EQW

To edit an equation with EQW, press \(\uparrow\) when the equation is on level 1 of the stack.

8.3.2 Modifications

To edit a variable name or a bigger expression with the command line editor, select it and press \(\text{\texttt{\textbf{\textbackslash EXIT}}}\).

It can also be edited with HP's EQUATION WRITER by typing \(\text{\texttt{\textbf{\textbackslash ENTER}}}\).

8.3.3 Temporary exit to the stack

It is possible to go temporarily to the stack by pressing \(\text{\texttt{\textbf{\textbackslash CONT}}}\). Press \(\text{\texttt{\textbf{\textbackslash ON}}}\) again to get back to EQW, in the same state as before halting.

8.3.4 Copy paste functions

Copy and paste functions are available. SUB copies the selected expression on the stack. REPL pastes the expression on level 1 either onto the selection, or where the edit cursor is. These commands are accessible in the menu or on keyboard:
8.3.5 Deletions
There are different ways to delete a part of an equation.
- In selection mode, typing numbers or characters suppresses the current selection. To enter a new expression that does not begin with a number or a name, pressing \hspace{1em} deletes the selection and goes into edit mode.
- To delete a one-argument function, just select this function and press \hspace{1em}.
- To delete one argument of a two-argument function, select it and press \hspace{1em}.
- Press \hspace{1em} to delete all arguments but the one selected.

8.3.6 Evaluations
Selected expressions can be processed by RPL commands \texttt{EVAL}, \texttt{COLCT} and \texttt{EXPAN}. The result replaces the old selection. These commands are accessible through menus or on keyboard:

\begin{itemize}
\item \texttt{EVAL}
\item \texttt{COLCT}
\item \texttt{EXPAN}
\end{itemize}

8.3.7 Custom program evaluation
An external program may be applied on the current selection. By pressing \texttt{\textit{CST}}, the program \texttt{STARTEQ} situated in the \texttt{HOMEDIR} is called, with the current selection on the stack level 1. This program must return the new expression on the stack level 1.

For example, the following program transforms all the INV functions by 1/:

\begin{verbatim}
\( \langle \langle \text{INV}('x') \text{ '1/' 'x' } \rangle \rangle \text{ MATCH DROP } \rangle
\end{verbatim}

8.4 Cursor mode

8.4.1 Switching to cursor mode
To get into cursor mode, press \hspace{1em}.

8.4.2 Movement
Arrow keys direct the cursor movement. The movement is faster when \hspace{1em} is held. To go directly to a border, hold \hspace{1em} and press one arrow key.

When the cursor stops, a box is drawn around the underlying expression.

8.4.3 Exiting cursor mode
There are two ways to return to selection mode.
\hspace{1em} returns to the same state prior to entering in cursor mode, whereas \hspace{1em} makes the expression under the cursor (which appears in a box) the new selection. This is a fast way to select a particular part of an equation.

8.5 Miscellaneous functions

8.5.1 Menu use
To show the menu, press a menu key, or \texttt{\textit{NXT}}. The menu may be hidden by pressing \hspace{1em}. When switching to cursor mode, the menu is automatically hidden.

8.5.2 Zoom
EQW can draw the equation using the system mini-font, by pressing \hspace{1em} or \hspace{1em}. This key can be used in any mode. A second keystroke returns to the stack font.

8.5.3 \texttt{→Grob}
The current selection may be pushed onto the stack as a graphic object by pressing \texttt{\textit{STD}} or \texttt{\textit{GRB}}. This function is the same as the RPL command \texttt{\textit{0 →GRB}}.

8.6 Error messages
An error message remains displayed as long as a key is pressed, to keep from destroying the expression.

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incomplete Expression</td>
<td>One cannot exit from EQW, or perform certain functions when the equation is not complete.</td>
</tr>
<tr>
<td>Global Expected</td>
<td>For some functions (e.g. differentiate, integrate), one of the arguments must be a variable name.</td>
</tr>
<tr>
<td>First Argument Missing</td>
<td>Two-argument functions must be typed after the first argument.</td>
</tr>
<tr>
<td>Two Arguments Expected \hspace{1em} \texttt{CLEAR}</td>
<td>(One argument delete) must be applied on a two-arguments or user function.</td>
</tr>
<tr>
<td>Edit Mode</td>
<td>RPL commands (EVAL, COLCT...) require selection mode.</td>
</tr>
<tr>
<td>1 Arg Function Needed \hspace{1em} \texttt{DEL}</td>
<td>must be used on a one-argument function like SIN.</td>
</tr>
</tbody>
</table>
8.7 Examples

\[ E = \int_0^1 |F(t)| dt \]

Press:

- \( E = \int_0^1 |F(t)| dt \)
- \( 0, 1/X \)
- \( \text{ABS} < \)
- \( F < T \)
- \( d T \)

\[ \delta(x) \frac{\cos x}{\sin^2 x} \]

Type the expression:

- \( \cos < x > \)
- \( / \sin < x > \)
- \( ^2 \)
- \( \delta \)

Save it on the stack:

- \( \text{MT} \) \( \text{N} \) \( \text{X} \) \( \text{SUB} \)

Differentiate it:

- \( \text{EVAL} \) \( \text{EVAL} \) \( \text{EVAL} \)

Simplify it:

- \( \text{EXP} \) \( \text{EXP} \) \( \text{Collect} \)

Rebuild the expression:

- \( \text{reverse} \) \( \text{DEL} \) \( \text{DEL} \)

Change \( INV \) into \( 1/ \) :

- \( \text{move on the last INV} \)

8.8 Key reference for EQW

8.8.1 Selection mode

<table>
<thead>
<tr>
<th>Inverted selection</th>
<th>Boxed selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{previous argument} )</td>
<td>( \text{previous variable or number} )</td>
</tr>
</tbody>
</table>
### Next argument (if on last argument, go to upper function).

### Next variable or number (if on last argument, go to upper function).

### Upper function.

### Inverted selection.

### First argument of the selected function.

### First argument.

### Last argument.

### Argument left roll.

### Argument right roll.

### Whole equation selection.

### First name or number of the selection, boxed selection.

### SUB, copies the selection to the stack.

### EXRND the selection.

### COLCTs the selection.

### EVALuates the selection.

### STO

### +GROB on the selection.

### Line command editing.

### HP’s Equation Writer editing.

### Adds/removes parentheses.

### One-argument function deletion.

### Argument deletion in a two-argument function or in a user function.

### Deletes all but the selection.

### Variable or name edition.

### Clears selection, goes into edit mode.

### Inserts a one-argument function.

### Inserts a function, the selection becomes the first argument (except for \( \text{                                                          } \) and \( \text{                                                          } \) where the selection becomes the main argument).

### Next argument, or create another argument in a user function.

### Creates a complex number, selection is the real part.

### Digit, character deletes the selection, starts editing.

### 8.8.2 Edit mode

### Digit, character Adds a character; a number followed by an alpha inserts an implicit multiply.

### Fast access to the character \( \text{                                                          } \) (not needed).

### Creates user function.

### Next argument, or creates another argument in a user function.

### 8.8.3 Selection and edit mode - common keys

### Saves changes and exits.

### Discards changes and exits.

### HRLT, temporary exit, a second \( \text{                                                          } \) returns to EQW, at the same place.

### OFF.

### Switches between stack font and little font.

### Display the first page of the menu.

### Menu, next page.

### Menu, previous page.

### Turn off the menu.

### REFLacrs the expression from the stack in the equation.

### 8.8.4 Cursor mode

### Movement.

### Fast movement.

### Border movement.

### Back in selection mode, no changes to the selection.

### Back in selection mode, the new selection is the part of the equation that is pointed to by the cursor.

### Discards changes and exits.
9 Matrix editor

The Matrix Editor is a matrix-oriented environment.

9.1 Presentation

The MK Matrix Editor can be used like the standard Matrix Writer, but it adds new features. Not only real and complex matrices can be edited, but also symbolic, character strings, program, etc., arrays.

For example, here is how the following symbolic matrix:

\[
\begin{bmatrix}
A+B & 1 & A \\
B & A-B & 2 \\
1 & B+C & C
\end{bmatrix}
\]

is represented on the stack as a list of lists:

\[
\langle \langle 'A+B' 1 'A' \rangle, \langle 'B' 'A-B' 2 \rangle, \langle 1 'B+C' 'C' \rangle \rangle
\]

To view and edit it under the Matrix Editor, press the down arrow or execute the command \( /c40/c39/c44/c55/c37 \).

The symbolic Matrix Writer is powerful enough to manage any kind of data. For example you can edit your contacts list.

With some external tools you can also transform the Symbolic Matrix Writer into a spreadsheet.

9.2 Usage

Two modes are available, one to create a new matrix, the other to edit an existing matrix.

To create a new matrix, press \( \downarrow \text{MATRX} \rightarrow \text{ENTER} \). An empty matrix is displayed, where any object can be entered. When \( \rightarrow \text{ENTER} \) is pressed, if all the elements in the matrix are of the same type, an array of that type is pushed on the stack (the content is displayed for real and complex numbers and strings), otherwise a list of lists is created (this is a multi-typed array).

In the second mode, when an existing array is edited, if a new element is entered outside the matrix, the other empty cells will be filled with "null" elements corresponding to the matrix type (0 for reals, "" for strings, NOVRL for multi-typed arrays, etc.).

PS: For some security reasons, only arrays of real and complex are created. If you want to create an array of another type, set the MK flag –27.

9.3 Reference

9.3.1 Keys

<table>
<thead>
<tr>
<th>Arrows ( \uparrow, \downarrow, \leftarrow, \rightarrow ) and ( \uparrow, \downarrow )</th>
<th>Movement.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrows with ( \uparrow, \downarrow )</td>
<td>Page by page movement.</td>
</tr>
<tr>
<td>Arrows with ( \leftarrow, \rightarrow )</td>
<td>Border movement.</td>
</tr>
<tr>
<td>( \text{DEL} )</td>
<td>Clears the current cell (fills it with the &quot;null&quot; element).</td>
</tr>
</tbody>
</table>

9.3.2 Menu keys

<table>
<thead>
<tr>
<th>EDIT</th>
<th>Edits the current cell in the command line editor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rightarrow \text{EDIT} )</td>
<td>Edits the current cell in the best environment (EDITB).</td>
</tr>
<tr>
<td>( \text{WID} )</td>
<td>Makes the cells narrower and displays one additional column.</td>
</tr>
<tr>
<td>( \text{WID+} )</td>
<td>Makes the cells wider and displays one less column.</td>
</tr>
<tr>
<td>( \text{GO+} \rightarrow \text{GO+} )</td>
<td>If the box is visible, makes the cursor move to the next column after entry (on by default).</td>
</tr>
<tr>
<td>( \text{GO+} \rightarrow \text{GO+} )</td>
<td>If the box is visible, makes the cursor move to the next row after entry.</td>
</tr>
<tr>
<td>( \text{FAST} \rightarrow \text{SLOW} )</td>
<td>Selects the cursor speed.</td>
</tr>
<tr>
<td>( \text{ROW} )</td>
<td>Inserts a row of &quot;null&quot; elements.</td>
</tr>
<tr>
<td>( \text{ROW} )</td>
<td>Deletes the current row.</td>
</tr>
<tr>
<td>( \text{COL} )</td>
<td>Inserts a column of &quot;null&quot; elements.</td>
</tr>
</tbody>
</table>
### 10 Grob editor (Picture2)

PICTURE2 creates and modifies graphic objects, similar to PICTURE, but is more powerful for bitmap creation. Grobs can be in two or four-level grayscale, and every drawing is made in real-time, with or without zoom.

#### 10.1 Usage

In this part, basis functions will be used to create a sample grob. See the reference part for a complete list of functions.

First create a four-level grayscale grob. Type `GREY 131 64 ENTER`

The blank grob is displayed on the stack.

Then edit it. Press ![down arrow]

There are two ways to use PICTURE2 functions.

- **Using the menu.** To display the menu, press ![left arrow] ![right arrow] and cycle through the menu pages.
- **With Shortcut keys.** Hold ![shift] and press the first letter of a function to launch it. When you don’t know the letter corresponding to a function, search the function in the menu, the shortcut letter is displayed bold.

Now draw an ellipse, using the menu. Press ![left arrow] to display the menu if necessary. The ellipse is called by the menu key ![ellipse]. Press ![left arrow] to hide the menu.

To call an ellipse with the shortcut keys, hold ![shift] and press ![e] (E for Ellipse, remember the bold E on the menu key).

The ellipse can be shaped, using the arrow keys.

Press ![enter] anytime to Zoom. Press ![enter] a second time to get back to the normal view.

Press ![up arrow] to display the cursors coordinates.

There are three coordinates shown: P1, P2 and P3. P1 is the current cursor position (it is underlined). To put the cursor on one coordinate, press ![1], ![2] or ![3]. In the case of the ellipse, P3 is the center of the ellipse, and P1 is a corner of the enclosing box. P2 is not used.

When the ellipse is correctly positioned (with P1 and P3), it is necessary to fix it on the grob. With no menu displayed, press:

- ![black] to fix it in white,
- ![light gray] to fix it in light gray (white for a two-level grayscale),
- ![dark gray] to fix it in dark gray (black for a two-level grayscale),
- ![black] to fix it in black,
Meta Kernel

Using the Meta Kernel

10.2 Reference

10.2.1 Commands

These commands can be typed in the command line, or called from RPL programs.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PICTURE2</td>
<td>The graphic editor which works with the PICT.</td>
</tr>
<tr>
<td>EDITB</td>
<td>The graphic editor which works with the grob on stack level 1.</td>
</tr>
<tr>
<td>PICT</td>
<td>Stores the grob on stack level 1 in the PICT (like PICT STO, but it also works the four-level grayscale grobs).</td>
</tr>
<tr>
<td>PICT+</td>
<td>Puts the current PICT grob on the stack.</td>
</tr>
<tr>
<td>GREY</td>
<td>Takes two B&amp;W grobs on the stack and creates one four-level grayscale grob with them. The grob on level 2 defines the low-weighted bits (a pixel in this grob becomes a light gray pixel in the gray-grob, a pixel in the grob on level 1 becomes a dark gray pixel).</td>
</tr>
<tr>
<td>GREY-</td>
<td>Equivalent to GREY+, but creates a four-level grayscale grob.</td>
</tr>
</tbody>
</table>
To convert a two-level grayscale grob into a four-level one, type `DUP +GREY`.

Contrary of `+GREY`, takes a four-level grayscale grob and returns two B&W grobs.

### 10.2.2 Keys

| ON | Exits PICTURE2. |
| OFF | Switches off the calculator. |
| + | Switches on or off the coordinates display. |
| - | Switches on or off the menu display. |
| 1 | Position the cursor at P1, and set P1 as the current cursor. |
| 1 | Stores the current cursor position in P1. |
| 2 | Position the cursor at P2. |
| 2 | Positions the cursor at P2, and set P2 as the current cursor. |
| 2 | Stores the current cursor position in P2. |
| 3 | Position the cursor at P3. |
| 3 | Positions the cursor at P3, and set P3 as the current cursor. |
| 3 | Stores the current cursor position in P3. |
| ENTER | Inverts the color of the pixel under the cursor. |
| ENTER | Copies P1 coordinates in P3. |
| ENTER | Switches between P1 and P3 as cursor (No effects if P2 is the current cursor). |
| ENTER | Puts the cursor coordinates on the stack, as a complex number. |
| ENTER | Puts the cursor coordinates on the stack, as a list of two binary numbers. |
| ← | Does a SUB. The grob boxed between P1 and P3 is put on the stack, as a grob. |
| CLEAR (DEL) | Cancels the current shape (line, circle...) and returns to normal cursor mode. |
| GREY | Clears the whole grob. |
| STO | Switches between the zoomed and the normal view. |
| ENTER | Puts the current grob on the stack. |
| Arrows | Cursor movement. |
| Arrows | Page by page cursor movement. |
| Arrows | Position the cursor on a border. |
| NXT | Menu keys. |
| A | In cursor mode, DOT+ in white. |
| A | In shape mode, fixes the shape in white. |
| B | In cursor mode, DOT+ in light gray (white in B&W). |
| B | In shape mode, fixes the shape in light gray. |
| C | In cursor mode, DOT+ in dark gray (black in B&W). |
| C | In shape mode, fixes the shape in dark gray. |
| D | In cursor mode, DOT+ in black. |
| D | In shape mode, fixes the shape in black. |
| E | In cursor mode, DOT+ in XOR mode. |
| E | In shape mode, fixes the shape in XOR mode. |
| F | Cancels the current shape and returns to normal cursor mode. |
| GROB | Puts a white pixel. |
| H | Puts a light gray pixel (white in B&W). |
| I | Puts a dark gray pixel (black in B&W). |
| J | Puts a black pixel. |

### 10.2.3 ALPHA keys

To use these keys, hold `α` down.

| 1 | Suspends the current editing, and edits the grob found on stack level 1 (Useful to edit a pattern for a pattern fill). Press `ON` to return to the other grob editing. |
| B | Box. P1 and P3 are two opposite corners. |
| C | Circle. P3 is the center, P1 is a point on the circle. |
| D | Executes the program stored in DOP I CT. |
| E | Ellipse. P3 is the center, P1 is a corner of the enclosing box. |
| F | Surface Fill. In B&W, inverses the surface color. In grayscale, the wanted color must be then defined with `αA`, `αB`, `αC`, or `αD`. Press `αE` or `←` to cancel. |
| G | Goes to the interactive stack. Press `ON` to return to PICTURE2. |
| H | Line. P1 and P3 are the ends. |
| I | Pattern fill (Motif). The pattern must be a 8x8 grob found on the stack level 1, in B&W or in grayscale. |
11 Filer

The Filer is a file utility that eases directory and port access.

Press \(\text{MEMORY} \rightarrow \text{VAR}\) to execute it. The first screen lets you choose the directory.

11.1 Ports and directories screen

The tree screen shows a list of ports and directories.
Press \(\uparrow, \downarrow\) to go throughout the list.
Press \(\text{ENTER} \uparrow, \text{OK} \downarrow\) to go in the selected location.
Press \(\text{ON} \uparrow, \text{CANCEL} \downarrow\) to exit to Filer.

11.2 File selection screen

On this screen are shown: the free memory, the total size of the selected files, a list of files and the menu.
Press \(\uparrow, \downarrow\) to move in the file list.
In each line, there are five fields: the selection number (if the object is selected), an icon depending on the file type, the name, the type and the size of the file.
Press \(\text{ENTER}\) to select or deselect a file.
Press \(\rightarrow\) on a directory or a library to go into it.
Press \(\text{LEFT}\) to go out of the current directory of library.
For all the commands that require a list of files, if there is no selection, the current line will be taken as the selection.
You can change the header in order to display the number of objects available and the working path (here port 0 home):

To hide the type and the size of every objects press \(\text{VAR}\)

11.3 Menu

Some commands work only in a special mode, for example only in the HOME directory. When you try to run these commands in an incorrect mode, a beep is produced.

| \(\text{EDIT}\) | Edits the current object. |
| \(\text{COPY}\) | Copies the selected objects. You define the destination with the tree screen: select the destination (a port or a directory) |
### 11.4 HEXA Editor

The screen shows 64 nibbles and their ASCII value. Each line represents 16 nibbles (separated by 8). On the bottom of the screen are written the assembly instruction, and the RPL object at the current address.

Press ▼ and ▲ to go one nibble backward or forward. Press △ and ▽ to move by 16 nibbles.

#### 11.4.1 Menu

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIND</td>
<td>Searches an hexadecimal string</td>
</tr>
<tr>
<td>NEXT</td>
<td>Searches the next occurrence.</td>
</tr>
<tr>
<td>GOTO</td>
<td>Goes to a given address.</td>
</tr>
<tr>
<td>POKE</td>
<td>Pushes the current object on the RPL stack. This command is very useful to recover data from a corrupted card (Invalid Card Data)</td>
</tr>
<tr>
<td>PANDA</td>
<td>Pushes the current address on the RPL stack as a binary integer</td>
</tr>
<tr>
<td>EXIT</td>
<td>Exits the Hexa Editor.</td>
</tr>
</tbody>
</table>

#### 11.4.2 Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXIT</td>
<td>Jumps to the next assembly instruction.</td>
</tr>
<tr>
<td>VAR</td>
<td>Jumps to the next RPL object.</td>
</tr>
<tr>
<td>EDIT</td>
<td>Goes in edit mode.</td>
</tr>
</tbody>
</table>
11.4.3 Edit mode

Very dangerous, use at your own risk!

Use the arrow keys to move, press a hexadecimal key from 0 to F to put the digit at the cursor position. Press \[ \text{Esc} \] \[ \text{Esc} \] again to exit the Edit Mode.
# Font Editor (EDF)

EDF is a font editor. It is accessed through the CHARS menu, where the current font can be edited. Any font can be edited by pushing it on the stack and pressing \( \triangleright \).

## 1.1 Font editing

When EDF is launched, scan mode is active, the font is displayed on the screen and the cursor is moved by the arrow keys. The current character number is displayed on the bottom of the screen (here 67: C).

- **MODIF**: Switches to edit mode. On the left of the screen, the character is displayed eight times bigger, on the right in its normal size.

- **SCAN**: Gets back to the CHARS mode.

The cursor is in the big character view, on the left. It is moved with the arrow keys, and \( \triangleright \) inverts the pixel under it.

## 1.2 CHARS mode

The chars mode is the default mode when EDF is launched. The cursor is moved by the arrow keys.

- **ECHON**: Pushes the current character on the command line. Press \( \text{ON} \) to exit.

- **ECHOP**: Pushes the current character on the command line and exits to the command line.

## 1.3 A session with CHARS

We will add a point inside the character C.

Press \( \triangleright \) to enter in CHARS. Use the arrows to move to the C character:

Use the arrows to move to the center of the character, press \( \triangleright \) to put a point:

Press \( \text{ON} \) to return to the stack display. See how the C characters have changed.
2 Machine Language Compiler (Masd)

2.1 Generalities on ML (Machine Language)

As the Saturn processor directly executes ML, the operating system cannot control what a ML program is doing.

On the HP 48 calculator, user data are stored in the same area as temporary data. When there is a bug in a ML program, you have best chance to lose your data's. So be very careful when programming in ML.

ML is a processor dependent language, so what you will learn on the HP 48 will not be useful on other processor. On the other hand, the programming techniques you will acquire are not dependent of the hardware and then will be reusable.

2.2 Launching Masd

To compile a program, put the string on the top of the stack and type RSM or use the RSM menu of the HPGK library.

2.3 Generalities on Masd Syntax

Masd expects a character string (called source) on the top of the stack.

A source is a set of instructions, comments, and separation characters and ends with a carriage return and an \ sign.

Masd is case sensitive, so be careful, as `boucle` and `BOUCLE` are two different labels.

Separation characters are those with an ASCII number below 32. They include spaces, tabs, line feed and carriage return.

Some instructions need a parameter, called field. Separation characters between an instruction and the field, are spaces, tabs, and points. Therefore R+R, R can be used instead of R+R, R.

Comments can be placed everywhere between two instructions. They begin with \ or / and finish at the end of the current line.

Directives change the way Masd interprets your source. These instructions begin with a $ and will be explained later.

While Masd is working, it displays messages on the screen to explain what is processed.

<table>
<thead>
<tr>
<th>Linking directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linking FileName</td>
<td>Compiles the FileName file. The main file is called Main.</td>
</tr>
<tr>
<td>Uplink</td>
<td>Ends the compilation of a file.</td>
</tr>
<tr>
<td>Compil. Glob</td>
<td>Compiles global jumps.</td>
</tr>
<tr>
<td>Compil. Exp</td>
<td>Calculates expressions.</td>
</tr>
<tr>
<td>Garbage</td>
<td>Reconfigures the memory.</td>
</tr>
<tr>
<td>Ghost LabelName</td>
<td>Detects an unused label.</td>
</tr>
</tbody>
</table>

If Masd detects one or more syntax errors, it will push a list describing all errors on the stack. This list is used by ER to find the location of errors, and let the user correct them.

Errors may be notified at the wrong place. For example if a label is not correctly defined, the errors will be located on the calling instructions.

2.4 Links

Links are source files that can be linked during compile time (equivalent to the {$I} directive in PASCAL and #include in C).

When a link call is encountered, Masd stops compiling the current link, compiles the new one and then continues compiling the first one.

Syntax:

`'FileName` links the file called FileName.

Note 1: A link can call other links.

Note 2: You can not use more than 256 links in your project.

Note 3: To know how Masd looks for files, see the File search section.

Note 4: Links are useful to cut projects in independent parts to allow fast and easy access to source code.

2.5 Using labels

A label is a marker in the program. The principal use of labels is to determine jump destinations.

A label is a set of less than 128 characters different from space, `+`, `-`, `*` and `/`. A label begins with a star `*` and ends with a separation character.

`*BigLoop` is the BigLoop label declaration.

Be careful about upper and lower cases!

Three types of labels can be used:
• Global labels
  A global label is a label that can be used everywhere in the project, like global variables in Pascal or C.

• Local labels
  A Local lab is a label that is only accessible in a local section like local variables in Pascal or C.
  A local section starts at the beginning of a source, after a global label, after a link (see link section) or after a \!LOCRL directive.
  A local section finishes at the end of a source, before a link, before a global label or before a \!LOCRL directive.
  A local label is identified by a ‘.’ as the first character.

• Link labels
  A link label is a label that exists only in the link where it is declared, like a private clause in Object Pascal.
  A link label is identified by a ‘-‘ as the first character.
  Note 1: In projects, using less global labels is better because a global label is longer to compile and because it gives a better program structure. A good habit is to use global labels to cut the program in subroutines, and to use local labels inside these subroutines.
  Note 2: The Meta Kernel command line is able to find labels in a source. Press blue shift \texttt{GOTO}.

2.6 Using constants

It is possible to define constants. It is useful to identify a memory address by a name, rather by the address itself.
For example, instead of typing \texttt{D1=80100} every time it is needed, it is better to declare \texttt{DC Result 80100} at the beginning of the project and then to type \texttt{D1=(5)Result} when needed.

Constant declaration:
\texttt{DCE CstName Expression or}
\texttt{EOUE CstName Expression or}
\texttt{DEFINE CstName Expression}
\texttt{DC CstName CstValue orEOU CstName CstValue}
\texttt{CstValue} is a hexadecimal number. A decimal number can be typed with a leading \# character.
\texttt{DC Foo 10} is same as \texttt{DC Foo #16}
Note 1: A constant cannot be given the same name as a declared label.
Note 2: The name of a constant follows the same rules as the name of a label.
Note 3: A constant value is always stored on 5 nibbles.

Masd introduces a ’programming register’ called CP (Constant Pointer) which helps to define constants. CP is defined by:
\texttt{CP=Cst or CPE=Expression}
CP is defined on 5 nibbles, its initial value is 80100.

\texttt{DCCP Increment CstName}
declares a constant with the current CP value and then increase CP by Increment.
Note: Increment is a hexadecimal value, to use a decimal value, put a leading \#.
For example, if CP equals to $10$
\texttt{DCCP 5 Foo}
defines a Foo constant with a value of $10$ and then change the value of CP to $15$.

Several constants can be defined, starting from CP.
\texttt{: Inc CstName0 CstName1 ... CstNameN-1 :}
defines \texttt{N} constants \texttt{CstName} with a value of \texttt{CP+$x^{*}$Inc} and then changes the CP value to \texttt{CP+$N^{*}$Inc}.
By default, \texttt{Inc} is a decimal number. It can be typed in hexadecimal, with a leading $\#$.

2.7 Expressions

An expression is a mathematical operation that is calculated while compiling.
Terms of this operation are hexadecimal or decimal values, constants or labels.
An expression stops on a separation character.
\texttt{DCCP 5 @Data}
... 
\texttt{D1=(5)@Data+$10/#2 D0=(5)$5+DUP LC(5)"DUP"+$5}
are correct expressions.
Notes:
• A hexadecimal value must begin with a $\#$ and a decimal value must begin with a $\#$.
• The $\#$ character equals to the address of the current instruction in absolute mode, or to the offset of the current instruction in standard mode (This value has no meaning in itself, but may be used to calculate the distance between a label and the current instruction).
• The value of a label is the address of the label in absolute mode, or the offset of the label in the program in normal mode (This value has no meaning in itself, but may be used to calculate the distance between a label and the current instruction).
• Entries from the STARTEXT table may be used. In an ambiguous case \(\text{DUP}+\#5\) may either be an addition \(\text{DUP} + 5\), or an entry \(\text{DUP}+\#5\), add \(\text{"DUP"}+\#5\).
• There is no priority (precedence) in operations. \(1+2*3=9\) instead of \(7\). Use parentheses to set priorities.
• You can’t use more than three level of parenthesis.
• Calculations are done on 64 bits.
• \(X\) divide by \(0 = \$FFFFFFFFFFFFFFF\).
• By default, expressions are calculated at the end of the compilation of the project.
• Masd can be forced to compile expressions using the \!COMPEXP directive. But be careful, if an expression uses a label that is not already declared, this will cause an error on the expression and not on the \!COMPEXP directive.
• Masd can be forced to compile the last expression using the \!COMPEXP directive with the same remarks than for \!COMPEXP. This can be useful when local labs are used in an expression.

2.8 Skips

Skips are a first step from ML to a third generation language, even if they are only another way to write SATURN instructions.

The basement of Skips is the Block structure.

A block is enclosed in \(<\text{and} \rangle\), and can be inside another block.

The following instructions deal with blocks.

<table>
<thead>
<tr>
<th>SKIPS instructions</th>
<th>Equivalents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKIP (\ldots) )</td>
<td>GOTO .S ... +.S</td>
</tr>
<tr>
<td>SKIPL (\ldots) )</td>
<td>GOTO0 .S ... +.S</td>
</tr>
<tr>
<td>SKIPC (\ldots) )</td>
<td>GOC .S ... +.S</td>
</tr>
<tr>
<td>SKC (\ldots) )</td>
<td>GOC .S ... +.S</td>
</tr>
<tr>
<td>SKIPNC (\ldots) )</td>
<td>GONC .S ... +.S</td>
</tr>
<tr>
<td>SKNC (\ldots) )</td>
<td>GONC .S ... +.S</td>
</tr>
<tr>
<td>TestSKIPYES (\ldots) )</td>
<td>TestGOYES .S ... +.S</td>
</tr>
<tr>
<td>Test (\ldots) )</td>
<td>TestGOYES .S ... +.S</td>
</tr>
<tr>
<td>Test+ (\ldots) )</td>
<td>/TestGOYES .S ... +.S</td>
</tr>
<tr>
<td>SKUB (\ldots) )</td>
<td>GOSUB .S ... +.S</td>
</tr>
<tr>
<td>SKUBL (\ldots) )</td>
<td>GOSUBL .S ... +.S</td>
</tr>
<tr>
<td>STRING (\ldots) )</td>
<td>#/02R2C G0IN5 *.S ... *.S (to create a character string)</td>
</tr>
<tr>
<td>CODE (\ldots) )</td>
<td>#/02DCC G0IN5 *.S ... *.S (to create a code object)</td>
</tr>
<tr>
<td>STROBJ PROLOG (\ldots) )</td>
<td>*/PROLOG G0IN5 *.S ... *.S (to create a ‘prolog’ length object)</td>
</tr>
</tbody>
</table>

/\text{Test} is the opposite of Test. For example if Test is ?A<C.A, /Test is ?A>=C.A. The test instructions dealing with the hardware register (?HST=0, ?MP=0, ?SR=0, ?X=M=0 and ?SB=0) cannot be inverted.

Once blocks are defined, special instructions can be used in them. These instructions called EXIT and UP allow to jump to the end or to the beginning of a block.

<table>
<thead>
<tr>
<th>These instructions</th>
<th>are equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;) )</td>
<td>*.Beginning</td>
</tr>
<tr>
<td>EXIT )</td>
<td>GOTO0.End</td>
</tr>
<tr>
<td>EXITC )</td>
<td>G0C.End</td>
</tr>
<tr>
<td>EXITNC )</td>
<td>G0C.End</td>
</tr>
<tr>
<td>?A=0,A EXIT )</td>
<td>?A=0,A \rightarrow.End</td>
</tr>
<tr>
<td>UP )</td>
<td>GOTO0.Beginning</td>
</tr>
<tr>
<td>UPC )</td>
<td>G0C.Beginning</td>
</tr>
<tr>
<td>UPNC )</td>
<td>G0C.Beginning</td>
</tr>
<tr>
<td>?A=0,A UP )</td>
<td>?A=0,A \rightarrow.Beginning</td>
</tr>
<tr>
<td>(*\text{.End}) )</td>
<td>*.End</td>
</tr>
</tbody>
</table>

Note: do not make confusion between EXIT and UP instructions, which are GOTOs, and EXIT and UP after a test, which are GOYES's.

EXIT and UP can jump to the beginning or to the end of an upper-level block by specifying the number of blocks to exit, after the UP or EXIT instructions.

<table>
<thead>
<tr>
<th>These instructions</th>
<th>Are equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;) ) &lt; ) &lt; )</td>
<td>*.Begin3</td>
</tr>
<tr>
<td>&lt; ) &lt; ) &lt; ) &lt; ) &lt; ) &lt; ) &lt; )</td>
<td>*.Begin2</td>
</tr>
<tr>
<td>&lt; ) &lt; ) &lt; ) &lt; ) &lt; ) &lt; ) &lt; )</td>
<td>*.Begin1</td>
</tr>
<tr>
<td>&lt; ) &lt; ) &lt; ) &lt; ) &lt; ) &lt; ) &lt; )</td>
<td>G0TO0.Beg2</td>
</tr>
<tr>
<td>&lt; ) &lt; ) &lt; ) &lt; ) &lt; ) &lt; ) &lt; )</td>
<td>GOTO0.Beg3</td>
</tr>
<tr>
<td>&lt; ) &lt; ) &lt; ) &lt; ) &lt; ) &lt; ) &lt; )</td>
<td>GOTO0.End1</td>
</tr>
<tr>
<td>&lt; ) &lt; ) &lt; ) &lt; ) &lt; ) &lt; ) &lt; )</td>
<td>GOTO0.End2</td>
</tr>
<tr>
<td>&lt; ) &lt; ) &lt; ) &lt; ) &lt; ) &lt; ) &lt; )</td>
<td>GOTO0.End3</td>
</tr>
</tbody>
</table>

Note: EXIT1 is equivalent to EXIT, and UP1 is equivalent to UP.

Using SKIPELSE, SKEC or SKENC instructions, two blocks create an IFNOT-THEN-ELSE structure.

<table>
<thead>
<tr>
<th>These instructions</th>
<th>Are equivalent to</th>
<th>Or in high-level language</th>
</tr>
</thead>
<tbody>
<tr>
<td>?A=0,A SKIPYES )</td>
<td>?A=0,A GOYES.Beg2</td>
<td>IF NOT A=0 THEN</td>
</tr>
</tbody>
</table>
2.9 Macros

If data are to be included in a project, they can be entered in a source file, using ".

But a simpler way is to include data from an external file, which is a macro. The macro file must be a character string, a graphic, a code object or a list.

In case of a string or a code, Masd includes only the data part (after the length)

In case of a graphic, only the graphic data will be included (no length, no dimensions).

In case of a list, only the first object of the list will be included following the previous rules.

The syntax is:

```
/FileName
```

Note: To know how Masd looks for the FileName file, see the following section.

2.10 Filename conventions

Masd sometimes needs to find a file in the HP 48 memory.

The file can be found either by specifying the complete file name and location, or only the file name to be search in the search path list.

The initial search path list contains the current directory, the upper directory and so on to the HOME directory.

Note: You can add a directory in the search path list using

```
INTERN: ReplName
```

where ReplName identifies a directory name using

the full pathname rules, explained below.

To specify a full path, use

```
H/ to specify HOMEDIR as the root.
```

```
*/x, where x is a port number, to specify a port as root.
```

This root is followed by a list of directories, ending with the name of the file.

```
4/TOTO/TITI/TUTU specifies the TUTU file in the TITI directory, stored in the TOTO backup of the fourth port.
```

```
H/ME/YOU specifies the YOU file in the ME directory, in the HOMEDIR.
```

2.11 Units

A unit is a part of a project, which can be compiled separately and stored in a file. The generated unit code can then be included in the project final code, without recompiling it.

A unit can use labels and constants defined in the calling program, and the calling program can use labels and constants defined in the unit.

Any source can be changed into a unit by placing a !UNIT directive in the source.

In the calling source file, type

```
\begin{verbatim}
\begin{verbatim}
```

Note 1: A unit is smaller in memory than the source file it comes from.

Note 2: Using a unit is a very fast operation, faster than compiling the source.

Note 3: Tools libraries can be created and distributed without sharing the source code.

2.12 Library creation

Libraries can be created with Masd using the following instructions:

```
LIB, MESSAGE, VISIBLE, HIDDEN and LCONFIG.
```

Here is a source producing a library:

```
!NO CODE
LIB "library name"
MESSAGE "library title"
1234 % message table
"Message 1"
"Message 2"
VISIBLE "VisibleName1"
"VisibleName2"
HIDDEN "HiddenName1"
"HiddenName2"
```

2.13 SysRPL mode

Masd can switch to SysRPL mode (also called System RPL or External) using the ! RPL directive. In RPL mode, lots of things are changing.

Separation characters don’t change (Characters of ASCII number below 32).
Comments begin with # and end at the end of the line, or begin with ‘<‘ and end with ‘>‘.

2.13.1 Instructions

In RPL mode, Masd interprets instructions in the following order.

a) Constants
If a constant exists with the same name, the constant value is used on 5 nibbles.
Example:
E0U TOTO 4E2CF
...
TOTO
will produce
FC2E4

b) External (in entry points table)
If an entry in the external table exists with the same name, the value associated with this entry is used.
BZUExt
will produce
DB10C
Note 1: Use the TableCreator program on your PC to create the table.
Note 2: This table can be stored anywhere in the HP 48 memory. Create a STARTEXT file in the HOMEDIR with the file name of the External table on the first line and a % on the second line.
Example of a STARTEXT file for a table stored in the Table backup to the fourth port:
"4/Table
@"
Note 3: Using an external table is much faster than using constants. On the other hand, constants are project dependant, which is not the case of an external table.

c) Tokens

<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>::</td>
<td>Program prolog $02D9D</td>
</tr>
<tr>
<td>;</td>
<td>List, Program or Algebraic end $0312B</td>
</tr>
<tr>
<td>(</td>
<td>List prolog $02A74</td>
</tr>
<tr>
<td>)</td>
<td>List end $0312B</td>
</tr>
<tr>
<td>SYMBOLIC</td>
<td>Algebraic prolog $02AB8</td>
</tr>
<tr>
<td>UNIT</td>
<td>Unit prolog $02ADA</td>
</tr>
<tr>
<td># cst</td>
<td>System Binary of cst value, given in hexadecimal.</td>
</tr>
<tr>
<td># cst</td>
<td>System Binary of cst value, given in decimal.</td>
</tr>
<tr>
<td># cst</td>
<td>System Binary (hexadecimal value). If the SB exists in ROM, its address is used.</td>
</tr>
<tr>
<td># cst</td>
<td>System Binary (decimal value). If the SB exists in ROM, its address is used.</td>
</tr>
<tr>
<td>PTR cst</td>
<td>Address.PTR 4E2CF generates FC2E4.</td>
</tr>
<tr>
<td>RCPTR cst1</td>
<td>Extended pointer.</td>
</tr>
<tr>
<td>RCPTR cst2</td>
<td>Extended pointer.</td>
</tr>
<tr>
<td>ROMPTR LibN ObjN</td>
<td>XLIB.</td>
</tr>
<tr>
<td>% real</td>
<td>Real number.</td>
</tr>
<tr>
<td>% real</td>
<td>Long real number.</td>
</tr>
<tr>
<td>% real1 real2</td>
<td>Complex number.</td>
</tr>
<tr>
<td>% real1 real2</td>
<td>Long complex number.</td>
</tr>
<tr>
<td>...</td>
<td>Character string. Special characters can be included by typing \ and the ASCII value on two hexadecimal characters.</td>
</tr>
<tr>
<td>ID name</td>
<td>Global name.</td>
</tr>
<tr>
<td>LAM name</td>
<td>Local name.</td>
</tr>
<tr>
<td>TAG chrs</td>
<td>Tagged object.</td>
</tr>
<tr>
<td>%xlibName</td>
<td>XLIB identified by its name. If it is a standard HP48 command (like &gt;DUP), the address is used instead of an XLIB object.</td>
</tr>
<tr>
<td>HXS Size Data</td>
<td>Binary integer ($02A4E), Size is in hexadecimal and Data is a set of hexadecimal characters. Example: HXS 5 FFR21</td>
</tr>
<tr>
<td>GROB Size Data</td>
<td>GROB ($02B1E).</td>
</tr>
<tr>
<td>LIBDRT Size Data</td>
<td>Library data ($02B88).</td>
</tr>
<tr>
<td>BAK Size Data</td>
<td>Backup ($02B62).</td>
</tr>
<tr>
<td>LIB Size Data</td>
<td>Library ($02B40).</td>
</tr>
<tr>
<td>EXT1 Size Data</td>
<td>Extended1 ($02BAA).</td>
</tr>
<tr>
<td>EXT2 Size Data</td>
<td>Extended2 ($02BCC).</td>
</tr>
<tr>
<td>EXT3 Size Data</td>
<td>Extended3 ($02BEE).</td>
</tr>
<tr>
<td>EXT4 Size Data</td>
<td>Extended4 ($02C01).</td>
</tr>
<tr>
<td>ARRAY Size Data</td>
<td>Array ($029E8).</td>
</tr>
<tr>
<td><strong>LNKARRAY</strong></td>
<td><strong>Size</strong></td>
</tr>
<tr>
<td>----------------</td>
<td>----------</td>
</tr>
<tr>
<td><strong>Data</strong></td>
<td>Linked Array ($02A0A).</td>
</tr>
<tr>
<td><strong>CODE</strong></td>
<td>Code object ($02DCC).</td>
</tr>
<tr>
<td><strong>NIBB</strong> Size</td>
<td>Includes directly hexadecimal data (no prolog).</td>
</tr>
<tr>
<td><strong>Data</strong></td>
<td>Includes the content of the file FileName.</td>
</tr>
<tr>
<td><strong>NIBHEX</strong> Data</td>
<td>Includes the source of the file FileName to be compiled (Like ' in ASM mode).</td>
</tr>
<tr>
<td><strong>CON</strong>(Size) <strong>Expr</strong></td>
<td>Defines a label (like * in ASM mode).</td>
</tr>
<tr>
<td><strong>CHR x</strong></td>
<td>Character object.</td>
</tr>
<tr>
<td><strong>INCLUDE</strong></td>
<td>Defines a constant (Like DCCP in ASM mode).</td>
</tr>
<tr>
<td><strong>FILE</strong> Name</td>
<td>Defines a constant (Like DC in ASM mode).</td>
</tr>
<tr>
<td><strong>EOU CstName</strong></td>
<td>Includes the content of the file FileName.</td>
</tr>
<tr>
<td><strong>Cst or</strong></td>
<td>Includes the source of the file FileName to be compiled (Like ' in ASM mode).</td>
</tr>
<tr>
<td><strong>EOUE</strong> CstName <strong>Expr</strong></td>
<td>Defines a label (like * in ASM mode).</td>
</tr>
<tr>
<td><strong>DEFINE</strong> CstName <strong>Expr</strong></td>
<td>Defines a constant (Like DCCP in ASM mode).</td>
</tr>
<tr>
<td><strong>EOUCP</strong> <strong>Interleave</strong> CstName</td>
<td>Note: A constant can be defined in ASM or RPL mode, and may be used in both modes.</td>
</tr>
<tr>
<td><strong>d)</strong> <strong>Decimal value (System Binary)</strong></td>
<td>d) Decimal value (System Binary)</td>
</tr>
<tr>
<td></td>
<td>If the instruction is not yet recognized, and if it is a decimal value, Masd generates a system binary.</td>
</tr>
<tr>
<td><strong>e)</strong> <strong>Unnamed local variables</strong></td>
<td>e) Unnamed local variables</td>
</tr>
<tr>
<td></td>
<td>Then, Masd tries to match the instruction with declared local variables.</td>
</tr>
<tr>
<td></td>
<td>A local environment is set using:</td>
</tr>
<tr>
<td></td>
<td>{{ var1 var2 ... varN }} with N&lt;23</td>
</tr>
<tr>
<td></td>
<td>These variables have names during compile time, but they are implemented as unnamed local variables, which are faster to access than named local variables.</td>
</tr>
<tr>
<td></td>
<td>A local variable is recalled by typing its name</td>
</tr>
<tr>
<td></td>
<td>Data can be stored in a local variable by typing its name, with a leading ! or =.</td>
</tr>
<tr>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td></td>
<td>{{ label1 label2 .. labelN }} will become:</td>
</tr>
<tr>
<td></td>
<td>' NULLLAM &lt;#&gt; NDUPN DOBIND (or IRLMBIND if there is only one variable)</td>
</tr>
<tr>
<td></td>
<td>And:</td>
</tr>
<tr>
<td></td>
<td>!label1 -&gt; IGETLAM</td>
</tr>
<tr>
<td></td>
<td>=label1 -&gt; IPUTLAM</td>
</tr>
<tr>
<td></td>
<td>!label1 -&gt; IPUTLAM</td>
</tr>
<tr>
<td></td>
<td>A local environment is not closed automatically, use ABND or other provided words.</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>{{ label1 label2 .. labelN }} will become:</td>
</tr>
<tr>
<td></td>
<td>' NULLLAM &lt;#&gt; NDUPN DOBIND (or IRLMBIND if there is only one variable)</td>
</tr>
<tr>
<td></td>
<td>And:</td>
</tr>
<tr>
<td></td>
<td>!label1 -&gt; IGETLAM</td>
</tr>
<tr>
<td></td>
<td>=label1 -&gt; IPUTLAM</td>
</tr>
<tr>
<td></td>
<td>!label1 -&gt; IPUTLAM</td>
</tr>
<tr>
<td><strong>Notes on RPL mode.</strong></td>
<td>Notes on RPL mode.</td>
</tr>
<tr>
<td>Masd switches back to ASM mode using the !ASM directive.</td>
<td></td>
</tr>
</tbody>
</table>

### 2.13.2 Special keystrokes in the command line

In the command line editor, some keystrokes help to enter characters that are specific to System RPL, and also to easily find a command (auto-completion).

- **Stat** followed by **.** displays a comma ,.

- **Stat** maintained and **.** displays a semicolon ;.

In alpha mode, **Stat** followed by **VAR** displays a vertical bar |.

In alpha mode, **Stat** maintained and **VAR** displays an anti-slash \.

Auto-completion works as follow:

Type the first characters of the command, for example DU.

Press **Stat** and maintain it. Press **SPC** once. The screen will now display all the commands that begin with DU:DUMP.

Use **Stat** and **SPC** to travel in the list.

Press **ENTER** to insert the command in the text.
To cancel the search without inserting a command, press \texttt{ON}.

**Inserting a bloc program:**

If no Command Line exists: create one with:

\begin{verbatim}
ASSEMBLY MODE:
"!NO CODE
!RPL
::
  ;
@"
SystemRPL MODE:
":="
  ;
@"
  \text{is the position of the cursor.}
\end{verbatim}

\begin{verbatim}
If a Command Line exits insert and indent a SystemRPL program bloc :
::
  ;
\end{verbatim}

### 2.13.3 Program example

To find or understand the syntax of a particular object, use \texttt{\*S2} to produce an \texttt{.mas} syntax-source.

As in ASM mode, a RPL source must end with a \texttt{\@}.

\begin{verbatim}
"!NO CODE
!RPL
::
  ONE \text{\langle My first program\rangle}
  !ASM
  \% Turn into ASM mode
  CODE \{ SAVE LOADRPL \}
  !RPL \{ Turn into SysRPL mode \}
  TWO
  ;
@"
\end{verbatim}

### 2.14 SATURN instructions syntax

In this section:

- \(x\) is an integer number between 1 and 16.
- \(h\) is a hexadecimal digit.
- \(a\) is a 1 to 16 or a 0 to 15 number depending on the current mode (0-15 or 1-16).
- \(f\) is a field \texttt{A}, \texttt{B}, \texttt{X}, \texttt{XS}, \texttt{P}, \texttt{WP}, \texttt{M} or \texttt{S}.
- \texttt{Reg} is a working register \texttt{A}, \texttt{B}, \texttt{C} or \texttt{D}.
- \texttt{SReg} is a save register \texttt{R0}, \texttt{R1}, \texttt{R2}, \texttt{R3} or \texttt{R4}.
- \texttt{Exp} is an expression.
- \texttt{Cst} is a constant. The value is given in hexadecimal or decimal using a leading \$ or \# respectively.
- \texttt{DRreg} is a pointer register \texttt{D0} or \texttt{D1}.
- \texttt{Data} is memory data pointed by \texttt{D0} or \texttt{D1}. It means \texttt{DRD0} or \texttt{DRD1}.

Note: For instructions that use two working registers, only the pairs \texttt{A-B}, \texttt{B-C}, \texttt{C-D} and \texttt{A-C} are available.

For instructions like \texttt{Reg1=Reg1}… you can write only \texttt{Reg1}… Example: \texttt{R=R+C.R} is the same as \texttt{A+C.R}.

### 2.14.1 Assigning 0 to a register

**Syntax:** \texttt{Reg=0.f}

**Example:** \texttt{R=0.M}

### 2.14.2 Loading a value in A or C

\texttt{LC} and \texttt{LA} instructions allow to load a constant value into \texttt{A} or \texttt{C} register.

- \texttt{LC \ hhh...hh} loads \texttt{x} nibbles into \texttt{C}.
- \texttt{LA \ hhh...hh} loads \texttt{x} nibbles into \texttt{A}.

**Example:** \texttt{LC 80180}

Note: \texttt{LC #12} allow to load 12 decimal into the 3 first nibbles of \texttt{C}. The number of nibbles used is the number of characters necessary to write the value (including the \#). So \texttt{#12} will take three nibbles.
LCRSC($x$) *Characters* loads the hexadecimal value of $x$ characters into C. $x$ must be between 1 and 8. LARSC($x$) if the counterpart for A.

Example: LCRSC(7) HP_MASD

LC($x$) Exp or LAR($x$) Exp load the result of an expression into C or A, using $x$ nibbles.

Example: LC(5) @Buffer+DataOffset

### 2.14.3 Loading a register value into another register

Syntax: `Reg1=Reg2.f`

Example: `R=E.X`

### 2.14.4 Exchange between two registers

Syntax: `Reg1Reg2.EX.f`

Example: `CDEX.W`

### 2.14.5 Addition

Syntax: `Reg1=Reg1+Reg2.f` or `Reg1+Reg2.f`

Example: `C+C.R.A` or `C+A.R`

Note: If `Reg1` and `Reg2` are same, this cause to multiply the register by two.

### 2.14.6 Subtraction

Syntax: `Reg1=Reg1-Reg2.f` or `Reg1-Reg2.f`

Example: `C=C.B.R` or `C.B.R`

Note: The following instructions are also available:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A=B-R$</td>
<td>$B=C-B$</td>
</tr>
<tr>
<td>$C=A-C$</td>
<td>$D=C-D$</td>
</tr>
</tbody>
</table>

### 2.14.7 Increment and decrement

Syntax: `Reg=Reg+Cst.f` or `Reg+Cst.f`


Note 1: The Saturn processor is not able to add a constant greater than 16 to a register but if `cst` is greater than 16, Masd will generate as many instructions as needed.

Note 2: Even if adding constants to a register is very useful, big constants should be avoided because this will slow down execution, and generate a big program.

Note 3: Adding a constant greater than 1 to a P, WP, XS or S field is a bugged SATURN instruction (problem with carry propagation). Use these instructions with care.

Note 4: After adding a constant greater than 16 to a register, the carry should not be tested.

### 2.14.8 Right nibbles shifting (divide by 16)

Syntax: `RegSR.f`

Example: `RSR.W`

### 2.14.9 Left nibbles shifting (multiply by 16)

Syntax: `RegSL.f`

Example: `RSL.W`

### 2.14.10 Right bit shifting (divide by 2)

Syntax: `RegSRB.f`

Example: `RSRB.W`

### 2.14.11 Right circular nibble shifting

Syntax: `RegSRC.f`

Example: `RSRC.W`

### 2.14.12 Left circular nibble shifting

Syntax: `RegSLC.f`

Example: `RSLC.W`

### 2.14.13 Logical AND

Syntax: `Reg1=Reg1&Reg2.f` or `Reg1&Reg2.f`

Example: `C=C.B.R` or `C&B.R`

### 2.14.14 Logical OR

Syntax: `Reg1=Reg1|Reg2.f` or `Reg1|Reg2.f`

Example: `C=C[B.R` or `C[B.R`

### 2.14.15 Logical NOT

Syntax: `Reg1=-Reg1-1.f`
2.14.16 Mathematical NOT
Syntax: Reg1 = Reg1.f
Example: C = ~C. R

2.14.17 Loading value into a R Register
Syntax: RRReg = Reg1
Example: R0 = R. W
Note: Reg can only be A or C.

2.14.18 Loading value into A or C from a R register
Syntax: Reg = RRReg1
Example: R = R1. X
Note: Reg can only be A or C.

2.14.19 Exchange between A or C and a R register
Syntax: RRRegEX
Example: R0=EX.R
Note: Reg can only be A or C.

2.14.20 Memory write (POKE)
These instructions write the value of A or C at the address pointed to by D0 or D1.
Syntax: Data = Reg.f or Data = Reg.x
Example: DRT1 = C.R or DRT0 = R. 10
Note: Reg can only be A or C.

2.14.21 Memory read (PEEK)
These instructions load into A or C the data pointed to by D0 or D1.
Syntax: Reg = Data.f or Reg = Data.x
Example: C = DRT1. R or R = DRT0. 10
Note: Reg can only be A or C.

2.14.22 D0 and D1 modifications
   a) Loading D0 and D1
   Syntax: DReg = hh or DReg = hhhh or DReg = hhhhh or
   DReg = (2) Exp or DReg = (4) Exp or DReg = (5) Exp
   Example: D0 = FF D0 = 12345  D1 = (5) toto + $5

   b) Exchanges between A or C and D0 or D1
   ♦ Loading A or C, field A, into D0 or D1
   Syntax: DReg = Reg
   Example: D0 = R
   Note: Reg can only be A or C.
   ♦ Loading the four low nibbles of A or C into D0 or D1
   Syntax: DReg = RegS
   Example: D0 = R8
   Note: Reg can only be A or C.
   ♦ Exchanging A or C, field A, and D0 or D1
   Syntax: RegDRegEX
   Example: R0=EX
   Note: Reg can only be A or C.
   ♦ Exchanging the 4 first nibbles of A or C and D0 or D1
   Syntax: RegDRegXS
   Example: R0=XS
   Note: Reg can only be A or C.

   c) Increment and decrement of D0 and D1
   Syntax: DReg = DReg + Cst or DReg + Cst
   Syntax: DReg = DReg - Cst or DReg - Cst
   Example: D0 = D0 + 12  D1 - 50
   Note 1: The Saturn processor is not able to add a constant greater than 16 to a register but if cst is greater than 16, Masd will generate as many instructions as needed.
2.14.23 Working registers tests

Notes:
- A test is always followed by RTNYES, GOYES, SKIPYES, EXIT, UP, GOTO, GOTOI, GOVLNG, GOSUB, GOSUBL or GOSBVSL.
- RTY is the same as RTNYES.
- An arrow (→) may be followed by a label name, then replacing GOYES, or may be followed by a skip block, which is equivalent to the inverse of the test followed by SKIPYES, to reproduce a IF-THEN structure. Example: ?A=C.R → ( ) is the same as R#C.R.
- SKIPYES may be omitted if followed by a skip block (↓).
- If the test if followed by a GOTO, GOTOI, GOVLNG, GOSUB, GOSUBL or GOSBVSL, Masd compiles the inverse of the test, to reproduce a GOYES with a larger range. Example: ?A=C.R GOTO B is the same as GOTO B.
- GOTO, GOTOI, GOVLNG, GOSUB, GOSUBL, GOSBVSL or → ( cannot be used after a HST test.
- A label name must follow a GOYES, GOTO, GOVLNG, GOSUB, GOSUBL or GOSBVSL.

a) Equality and inequality tests
Syntax: ?Reg1=Reg2.f ?Reg1≠Reg2.f
Example: ?A=C.B ≠C#D.A
Note: The HP inequality character may be used.

b) Lower and greater tests
Syntax: ?Reg1<Reg2.f ?Reg1≥Reg2.f
Example: ?C>B ≥C#E.D.A
Note: The HP lower or equal and greater or equal characters may be used.

c) Nullity tests
Syntax: ?Reg=0.f ?Reg≠0.f
Example: ?A=0.B ≠C#0.XS
Note: The HP inequality character may be used.

2.14.24 Working with some bits of A or C register

?RegBIT=v.a ?RegBIT=v.a where Reg is A or C, v is 0 or 1 (reset or set), and a is the bit number.
Examples: ABIT=0.5, CBIT=1.3 GOYES TOTO

2.14.25 Operations on PC

A=PC C=PC PC=A PC=C RPCEX CPCEX PC=(A) PC=(C)

2.14.26 Working with the Hardware Status Register

SB=0 XM=0 SR=0 MP=0 HST=a
?SB=0 ?XM=0 ?SR=0 ?MP=0 HST=a

2.14.27 Working with P

P=a
P=P+1 P+1 P=P-1 P-1
?P=a ≠P#a
P=C,a C=P,a CPEX,a
C=C+P+1 C+P+1

2.14.28 Jump instructions

GOTO label
GOTOL label or GOLONG label
GOVLNG Cst Cst is an hexadecimal number.
GOVLNG =label label is a constant, or a label in absolute mode
GOVLNG ="COMMAND" Command is an entry in the STARTEXT table.
GOSUB label
GOSUBL label
GOSBVBL Cst Cst is a hexadecimal number.
GOSBVBL =label label is a constant, or a label in absolute mode.
GOSBVBL ="COMMAND" COMMAND is an entry in the STARTEXT table.

GOC label
GONC label
GOTOC label same as SKIPNC GOTO label
GOTONC label same as SKIPC GOTO label

RTN RTNSXM RTNCC RTNSC RTI
2.14.29 Exchanges between C and RSTK

C⇒RSTK and RSTK⇒C instructions allow to push to or pop data from the Saturn return stack.

2.14.30 Input / output instructions

- OUT=C,OUT=C,R=IN and C=IN
  - Note 1: R=IN and C=IN instructions are bugged (they only work on even addresses). So use R=IN2 and C=IN2, which are ROM calls to R=IN and C=IN instructions.
  - Note 2: if the beginning of ROM is not usable (because it is recovered by RAM), use R=IN3 and C=IN3, which are calls to R=IN and C=IN instructions in the Meta Kernel Card.
  - Note 3: OUT=C=IN is a ROM call that does OUT=C C=IN.
  - Note 4: OUT=C=IN3 is the same, but in the Meta Kernel Card (works even if lower ROM is recovered).

2.14.31 Processor control instructions

- Working mode modification
- SETDEC SETHEX
- other instructions
- UNCNQF CONFIG RESET C=ID
- SHUTDN INTOF INTERT RSI
- SREQ?
- BUSCB BUSCC BUSCD

2.14.32 New instructions of Masd

<table>
<thead>
<tr>
<th>G0INC label</th>
<th>Equivalent to LC(5) label-8. (&amp; is the address of the instruction)</th>
</tr>
</thead>
<tbody>
<tr>
<td>G0INR label</td>
<td>Equivalent to LR(5) label-8. (&amp; is the address of the instruction)</td>
</tr>
<tr>
<td>&lt; and &gt;</td>
<td>Program prolog 02D90 and epilog 0312B.</td>
</tr>
<tr>
<td>-</td>
<td>Noeval</td>
</tr>
<tr>
<td>Rlabel</td>
<td>In absolute mode, places the address of the label.</td>
</tr>
<tr>
<td>Blabel</td>
<td>In absolute mode, places the address of the label+5.</td>
</tr>
<tr>
<td>$hhhh..hhhh</td>
<td>Includes hexadecimal data. Example: $12RCD545680B.</td>
</tr>
<tr>
<td>/hhhh..hhhh</td>
<td>Includes hexadecimal data in reverse order. Example: /123ABC is equivalent to $CBR321.</td>
</tr>
<tr>
<td>(x)Exp or</td>
<td>Places the value of Exp in the code, on x nibbles.</td>
</tr>
<tr>
<td>CON(x)Exp or</td>
<td>acquainted with EXP(x)EXP.</td>
</tr>
<tr>
<td>ASCII</td>
<td>Includes ASCII data. The end of the string is the next + or carriage return. Example: +Hello+. To output a + character, put it twice.</td>
</tr>
<tr>
<td>INCLUDE File</td>
<td>Includes directly the specified file as data.</td>
</tr>
<tr>
<td>eCst</td>
<td>Includes the value of Cst on 5 nibbles. e123 is equivalent to $/00123.</td>
</tr>
<tr>
<td>cCst</td>
<td>Includes a system binary of value Cst.</td>
</tr>
<tr>
<td>G0IN5 lab G5 lab</td>
<td>Same as $(x)label-8. with x=5, 4, 3 or 2. Useful to create a jump table.</td>
</tr>
<tr>
<td>G0IN4 lab G4 lab</td>
<td></td>
</tr>
<tr>
<td>G0IN3 lab G3 lab</td>
<td></td>
</tr>
<tr>
<td>G0IN2 lab G2 lab</td>
<td></td>
</tr>
<tr>
<td>SAVE</td>
<td>Equivalent to GOSBVL 0679B</td>
</tr>
<tr>
<td>LOAD</td>
<td>Equivalent to GOSBVL 06702</td>
</tr>
<tr>
<td>RPL</td>
<td>Equivalent to R=DR0.R D0+5 PC=(R)</td>
</tr>
<tr>
<td>LORPL</td>
<td>Equivalent to GOVLHG 138B9</td>
</tr>
<tr>
<td>INTOFF2</td>
<td>Equivalent to GOSBVL 01115</td>
</tr>
<tr>
<td>INTON2</td>
<td>Equivalent to GOSBVL 010E5</td>
</tr>
<tr>
<td>ERROR .C</td>
<td>Equivalent to GOSBVL 10FB0</td>
</tr>
<tr>
<td>RES .STR</td>
<td>Equivalent to GOSBVL 05B7D</td>
</tr>
<tr>
<td>RES .ROOM</td>
<td>Equivalent to GOSBVL 039BE</td>
</tr>
<tr>
<td>[ ... ] Size</td>
<td>Makes the code inside the brackets fit the given size, by adding zeros if necessary. Only three levels of brackets can be used.</td>
</tr>
</tbody>
</table>

2.15 Masd directives

| IPATH+ DirName | Add the specified directory in the search path list.          |
| LINK chr       | Change the char identifying link labs to chr.                 |
| !OFF           | Shut down the screen. This speeds up the HP48 by 13%.         |
| !LOCAL chr     | Change the char identifying local labs to chr.                 |
| !DISP message  | Display the message on the screen.                           |
| !UNIT          | The output file will be a unit.                               |
| !WAROFF        | Masd will not display ghost labels.                           |
| !NO CODE       | Masd will not generate a $02DCC prolog but will directly output the data. |
| !1-16          | Switch to 1-16 mode.                                          |
| !1-15          | Switch to 0-15 mode.                                          |
| !RPL           | Switch to RPL mode.                                           |
| !ASM           | Switch to ASM mode.                                           |
| !FL=0.a        | Clear the a compilation flag.                                |

RTNC RTNHC
RTNYES or RTY after a test.
2.16 Error messages

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid File</td>
<td>The file is not a source or a macro. (must end with a @)</td>
</tr>
<tr>
<td>Too many links</td>
<td>Only 256 links are supported. May be due to a recursion.</td>
</tr>
<tr>
<td>Unknown Instr.</td>
<td>Unknown instruction.</td>
</tr>
<tr>
<td>Bad Field</td>
<td>Incorrect field.</td>
</tr>
<tr>
<td>B-15 Expected</td>
<td>An integer between 0 and 15 is expected.</td>
</tr>
<tr>
<td>I-16 Expected</td>
<td>An integer between 1 and 16 is expected.</td>
</tr>
<tr>
<td>Error Stack Size</td>
<td>You use more than three brackets.</td>
</tr>
<tr>
<td>Size Too Small</td>
<td>Program between brackets is too big.</td>
</tr>
<tr>
<td>Close Size</td>
<td>There are more ] than [.</td>
</tr>
<tr>
<td>Label Expected</td>
<td>A label name is expected.</td>
</tr>
<tr>
<td>Const Expected</td>
<td>A hexadecimal number is expected.</td>
</tr>
<tr>
<td>Can’t Find</td>
<td>Can’t find the label or the file.</td>
</tr>
<tr>
<td>Lab. Already Dec.</td>
<td>A label or a constant is already declared with the same name.</td>
</tr>
<tr>
<td>Need (</td>
<td>An opening ( was expected.</td>
</tr>
<tr>
<td>Need )</td>
<td>A closing ) was expected.</td>
</tr>
<tr>
<td>Forbidden</td>
<td>Operation not allowed.</td>
</tr>
<tr>
<td>Bad Expression</td>
<td>Error in expression.</td>
</tr>
<tr>
<td>Jump too Long</td>
<td>No comments.</td>
</tr>
</tbody>
</table>

3 Disassembler

The disassembler converts binary code into a source string. There are two ways to use it: with a Code on the stack level 1 or with two addresses (in binary) that point to the beginning and the end of the code to be disassembled.

The syntax used is Masd syntax, in mode 0-15.

Each line contains an address and an instruction.

If the MK system flag -7 is set (with -7 SF2), addresses are not shown, except for the destinations of jumps. In this case, the resulting source may be then reassembled if needed.

Example:

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>AE734</td>
<td>GOSBVL 06798</td>
</tr>
<tr>
<td>AE738</td>
<td>LC 01000</td>
</tr>
<tr>
<td>AE742</td>
<td>C-R A</td>
</tr>
<tr>
<td>AE744</td>
<td>GONC AE742</td>
</tr>
<tr>
<td>AE747</td>
<td>GOVLNG 13889</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>RE734</td>
<td>GOSBVL 06798</td>
</tr>
<tr>
<td>RE738</td>
<td>LC 01000</td>
</tr>
<tr>
<td>RE742</td>
<td>*RE742</td>
</tr>
<tr>
<td>RE744</td>
<td>GONC RE742</td>
</tr>
<tr>
<td>RE747</td>
<td>GOVLNG 13889</td>
</tr>
</tbody>
</table>

4 Miscellaneous Utilities

These utilities are found in the MDGKER library.

Some of the following commands are marked **Dangerous!**, which means that incorrect use of them can corrupt memory. Use them at your own risk!

4.1 →H

Returns a character string containing the hexadecimal numbers that form the object on stack level 1 in memory.

Example: ‘R’ →H returns "84E201014"

4.2 H→

Inverse of →H, it converts a hexadecimal character string to the objects coded in it. **Dangerous!**

Example: "47A209C2A2B2130" H→ returns (1 )

4.3 S→H

Converts a character string into its hexadecimal form (every character is converted to two characters).

Example: "HI" S→H returns "8494"
4.4 H→S \[H\leftrightarrow S\]
Inverse of S→H. Every couple of hexadecimal characters gives an ASCII character.
Example: "8405" H→S returns "HP"

4.5 CD→ CD+1
Does a →H on the content of a Code object.

4.6 --→CD CD
Inverse of CD→. A hexadecimal string is placed inside a Code object.

4.7 --→A A
Returns the memory address of the object on stack level 1.
Example: 1 +A returns #2R29h (address of the real number 1)

4.8 A→ A
Puts the object pointed to by the given address on stack. Dangerous!
Example: #1B4Rh A→ returns SIN

4.9 PEEK PEEK
Reads memory. Level 2 contains the starting address (a binary number), level 1 gives the number of nibbles to be read (a binary number). A hexadecimal string is returned on the stack.
Example: #0 #10d PEEK returns "2369B1080R"

4.10 POKE POKE
Writes memory. Level 1 contains the hexadecimal string, level 1 the start address. Dangerous!
Example: #101h "C" POKE changes the contrast.

4.11 SREV SREV
Reverses the order of the characters in a string.
Example: "HP 48" SREV returns "84 PH"

4.12 PRG~LST PRG~LST
Transforms a program to a list and vice-versa.
Example: ( SWAP DROP ) PRG~LST returns PRG SWAP DROP END.

4.13 --→RAM --→RAM
Creates a new copy of the object in memory, equivalent to NEWOB, but it also works with ROM objects.
Example: ~ DUP --RAM returns PRG e 16RA5 e 03198 END (the DUP program in ROM).

4.14 Apeek APEEK
Reads the address at a given address (binary number on stack level 1).
Example: #3188h APEEK returns #3180h

4.15 R~SB R~SB
Transforms a real number to a system binary and vice-versa.
Example: 32 R~SB returns R 20h

4.16 --→GROB2 --→GROB
Transforms an object to its graphical representation.
The object is on stack level 2. A real number at level 1 gives the desired height:
- 1: Mini-font
- 2: Stack font
- 3: Big font
• 0: Big font, but if the object is an algebraic, the **MK** equation editor is used to draw it.

If the system flag2 -6 is set, the carriage returns in a string are used to draw multiple lines, otherwise if this flag -6 is cleared, the string will only be drawn on one line.

Example: `'\text{SIN(X)}'` 0 \text{GR0B2} returns \text{Graphic} 31 \times 9

4.17 \rightarrow \text{HEADER} \text{\texttt{\#CLAIM}}

Defines the number of lines of the status area, 0, 1 or 2.

4.18 \text{HEADER} \rightarrow \text{HEAOE}

Returns the current status area height.

4.19 \text{INPUT2} \text{\texttt{INPUT}}

Works like \text{INPUT}, but uses the **Meta Kernel** environment.

4.20 \text{DISP2} \text{\texttt{DISP}}

Works like \text{DISP}, but uses the current system font (defined with \texttt{\#FONT}), so that up to 9 lines can be written.

4.21 \text{HALT2} \text{\texttt{HALT}}

Works like \text{HALT}, but returns to the **Meta Kernel** stack (\texttt{HALT} returns to the standard HP stack).

\text{\textbf{Note:}} \text{DEBUG, SST and NEXT} in the PRG-RUN menu have been rewritten to work under the **Meta Kernel**.

4.22 \rightarrow \text{S2} \text{\texttt{S2}}

Converts an object into a character string in Masd format, in RPL mode.

4.23 \rightarrow \text{S4} \text{\texttt{S4}}

Converts an object into a character string, like the standard \texttt{S4}, \texttt{EDT} uses it.

4.24 \text{CHOOSE2} \text{\texttt{CHOOSE}}

Works like \text{CHOOSE}, but uses the current system font.

4.25 \text{EDIT} \text{\texttt{EDIT}}

Starts editing the object on stack level 1, with the command line editor.

4.26 \text{VISIT} \text{\texttt{VISIT}}

Starts editing the object stored in the given variable name, with the command line editor.

4.27 \text{EDITB} \text{\texttt{EDIT}}

Equivalent to \text{EDIT}, but uses the best adapted environment (command line, equation editor, graphic editor...), depending on the object type.

4.28 \text{VISITB} \text{\texttt{VISIT}}

Equivalent to \text{VISIT}, but uses the best adapted environment (command line, equation editor, graphic editor...), depending on the object type.

4.29 \text{EQW} \text{\texttt{EQW}}

Launches the equation editor with the expression given on stack level 1.

4.30 \text{FILER} \text{\texttt{FILER}}

Launches the filer.

4.31 \text{ASM} \text{\texttt{ASM}}

Launches the assembler with the source string on stack.
4.32 ER Edits the error after assembling. There must be a string at level 2 and the errors list at level 1.

4.33 ASM→ Starts the disassembler. Consult Page 43.

4.34 →NDISP Defines the maximum number of lines taken by an object displayed on stack (including grobs and algebraics).

4.35 →FONT Sets the font object on stack level 1 as the system font.

4.36 FONT→ Returns the current system font on the stack.

4.37 FONT8, FONT6 Returns the standard Meta Kernel fonts 6 and 8.

4.38 KERNEL? Returns nothing. It is used to test if the Meta Kernel is present, if a program needs it.


4.40 SREPL Does a find/replace in a character string. The main string is at level 3, the find string at level 2 and the replace string at level 1. Every occurrence of the find string in the main string will be replaced by the replace string. The find and replace strings may have different lengths.

4.41 AR~LST Converts an array into a list of lists and vice-versa.

4.42 DIMS Puts the size of the array contained in a list of lists on stack level 2 and 1 on stack level one. If it is not an array, returns 0.

4.43 →MINIFONT Sets the mini-font object on stack level 1 as the system mini-font.

4.44 MINIFONT→ Returns the current system mini-font on the stack.

4.45 FNT2GRB.PRG Converts an MK font to a grob font (common 6,7 or 8 x 2048 grob). This program is on the floppy disk.
4.46 GRB2FNT.PRG

Converts a grob font to an MK font. The grob in at level 3, the name is a character string at level 2 and the identifier a real number at level 1.

This program is on the floppy disk.
Appendixes

1 Introduction to assembly language
   [in complete documentation]

2 Introduction to System RPL
   [in complete documentation]

3 Tips and tricks

3.1 Meta Kernel and UFL

UFL (Universal Font Library) is an attempt to normalize the use of fonts in HP48 programs. The idea is to allow all programs to share the same fonts, instead of having as many fonts as software. UFL is a standard library. It provides two different fonts, the normal font or the mini-font. The Meta Kernel is able to work only with the last one, since the MK provides many more powerful features in font management, such as more than 240 different fonts at the same time, while UFL can manage only one. Also, you can change the system font, without restarting your HP48 with an ON-C.

You will find more information about the UFL at this URL:
http://www.engr.uvic.ca/~aschoorl/ufl/

The Meta Kernel uses its own mini-font, but may use the UFL mini-font. Just execute these few lines, or add them in the STARTUP file:

```
1 FNT →MINIFONT
```

3.2 STARTED and EXITED examples

3.2.1 Remove the header display

When you edit files, it may be useful to have the whole screen available. Using the STARTED and EXITED capabilities, it is very easy to remove the header while editing, and restore it after.

```
STARTED:
<
  0 →HEADER
>

EXITED:
<
  2 →HEADER
>
```

3.2.2 Edit compressed files

When you are short in memory, it may be useful to compress some files. BZ was written by Mika Heiskanen (http://www.hut.fi/~mheiskan/) it is a very efficient and very fast compressor.

You will find the BZPlus package on http://www.hpcalc.org/utils/compress/.

The problem with the compressor, is that if you want to edit the file, you must uncompress the file manually, edit it, compress it again, then store it.

With STARTED and EXITED, you won't have to do that anymore. The idea is to check if a file is a compressed file with STARTED, if yes uncompress it, and when leaving the command line, compress it with EXITED.

The main difficulty is that the Meta Kernel can edit more than one text at the same time. Then you must save the history of the edited files, in order to know if the file was compressed or not. The history will be saved in the HOME directory with the name CHECK.BZ

STARTED is called by the Meta Kernel just before editing an object, and finds the object on the first stack level. Sometimes the MK calls STARTED without any valid object, in this case, you find the object NULL$ on the stack.

EXITED is called by the Meta Kernel just after exiting the command line by pressing ENTER or ON. When the user press ENTER, the MK pushes two objects on the stacks as follow:

```
2: Object
1: TRUE
```

If the user has pressed ON, only FALSE is pushed on the first stack level.

The following STARTED and EXITED use both System RPL and User RPL, they may be enhanced very easily by using only System RPL. Anyway, it is a good example of how to use the MASD in RPL mode.

```
" !NO CODE
!RPL ::
  «DUP NULL$ EQ ?SKIP » (Check if it’s the NULL$ object)
```

If yes do nothing)
3.3 Meta Kernel and Erable

Erable is a very powerful Algebra system for the HP48GX. Combined with the Meta Kernel, you will get one of the best handheld computer algebra system (far better than the TI-92). Here is an example of an STARTEQW file, which allow to use all Erable’s functions from the Equation Writer, by pressing [ST]. This STARTEQW was written by Erable’s author: Bernard Parisse

You will find more information about Erable at:

STARTEQW:
"ERABLE"
( "1.SIMP" "2.TRIG" "3.INTDER" "4.EXEC"
"5.PLOT" "6.CFG" "7.ARIT" )
1 CHOOSE2 IF THEN
DUP 
( EXPA COLC EXPLIN SINCOS TEXPA LNCOLC TSIMP
( "0.BACK" STARTEQW ) )
( TRIGLN TRIGCOS TRIGSIN TAN2SC TAN2SC2 HALFTAN
( "0.BACK" STARTEQW ) )
( der1 RISCH PF LAP ILAP ( "0.BACK" STARTEQW ) )
( "1.NEW VX" ( 'DUP 'VX' STO » )
( "2.REPLACE" "REPLACE" ( "X=" ALG V )
INPUT2 STR+ EXEC » )
( "3.COMMAND" "COMMAND" ( "" ALG V )
INPUT2 STR+ EVAL » )
( "4.ISOL VX" VX EXEC » )
( "0.BACK" STARTEQW )
)
( "1.ERASE AND PLOT" "ERASE DUP STEQ
# B4045h LIBEVAL » )
( "2.ADD TO PLOT" ( #0 DUP TYPE 5 ==
IF THEN OVER + ELSE OVER 2 +LIST END
STEQ # B4045h LIBEVAL » )
( "3.PLOT MENU" ( # B4045h LIBEVAL »
( "0.BACK" STARTEQW )
)
)
3.3.1 STARTOFF and TOFF example

Here is a completely useless program. But like all completely useless programs, it is completely essential.

This is a screen saver program, it will display points in the screen until you press a key.

If you want the Meta Kernel to run this program after two minutes of idle time, put #983040d in the 'TOFF' file.

```
STARTOFF:
  PICT+ LCD+ ->PICT ( #0 #0 ) PVIEW
DO
  RAND 131 * R+B
  RAND 64 * R+B
  2 ->LIST PIXON
UNTIL KEY END DROP
TEXT +PICT
```

4 The MK for Programmers

4.1 Customizing the Filer

4.1.1 Overview

The filer is completely customizable. You can run your own programs from inside the filer. These programs can utilize all of the functionality of the filer, which the internal functions use. You can also access all internal functions. This customization is accomplished using a custom menu that displays labels for each of your custom programs. In the filer, if you press CST or CUSTOM in the built-in menu, the filer will try to find the file 'FILER.CUSTOM'. This file is used to create your custom menu.

4.1.2 FILER.CUSTOM Format

FILER.CUSTOM is a list with the following arguments:

\[ < \]

\[ GROB \ 131* n \]

This is the GROB that will be displayed in the menu area when you press \( \text{CST} \). You can also use a program that gives a GROB as a result.

\[ \text{System Binary 1} \]

This is an integer that tells the filer how many pages the menu will use. For example, if you want two pages (12 labels) then GROB 131\*n would be a 131*16 GROB and SB1 would be 2.

\[ \text{System Binary 2} \]

This integer allows you to control when the program will be started. There are 5 possibilities:

- 0 You can run the program everywhere (VAR, PORT, BACKUP, LIB)
- 1 Only when you are in VAR
- 2 Won't run if you are in a library
- 3 Won't run if you are in a backup
- 4 Will run only in port (home of the port)

\[ \text{System Binary 3} \]

This integer tells the filer how the associated program will be run. This also allows you to access all the filer's internal subroutines. There are two categories of custom programs: internal calls, and custom calls.

\[ \text{System Binary 4} \]

This is a shortcut-key for the program.

\[ ....An entry just as above for each custom program \]

4.1.3 Explanation of each component

a) System Binary 1 and GROB

This is an integer that tells the FILER how many pages the menu will use. For example, if you want two pages (12 labels) then GROB 131\*n would be a 131*16 GROB and SB1 would be 2.

b) System binary 2

This integer allows you to control when the program will be started.

There are 5 possibilities:

<table>
<thead>
<tr>
<th>SB</th>
<th>Program Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>You can run the program everywhere (VAR, PORT, BACKUP, LIB)</td>
</tr>
<tr>
<td>1</td>
<td>Only when you are in VAR</td>
</tr>
<tr>
<td>2</td>
<td>Won’t run if you are in a library</td>
</tr>
<tr>
<td>3</td>
<td>Won’t run if you are in a backup</td>
</tr>
<tr>
<td>4</td>
<td>Will run only in port (home of the port)</td>
</tr>
</tbody>
</table>

c) System binary 3

This integer tells the filer how the associated program will be run. This also allows you to access all the filer's internal subroutines. There are two categories of custom programs: internal calls, and custom calls.

NOTE: You can use a Binary integer instead of System Binary (it is easier to manage with the User-RPL compiler).
d) Internal calls

The internal calls are included so that you can access the built-in functionalities of the filer from inside your custom menu. For these calls, you don’t need to put a program in the file unless you want to allow a shortkey (Ex Left-Shift 2). If you use a shortkey then add any program you want because it won’t be run at all. You should use the smallest possible program (ex: TakeOver $40788) internal empty program.

<table>
<thead>
<tr>
<th>SB</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Bip</td>
</tr>
<tr>
<td>1</td>
<td>Info</td>
</tr>
<tr>
<td>2</td>
<td>Hexa</td>
</tr>
<tr>
<td>3</td>
<td>View</td>
</tr>
<tr>
<td>4</td>
<td>Arbo</td>
</tr>
<tr>
<td>5</td>
<td>Up</td>
</tr>
<tr>
<td>6</td>
<td>MaxUp</td>
</tr>
<tr>
<td>7</td>
<td>Down</td>
</tr>
<tr>
<td>8</td>
<td>Maxdown</td>
</tr>
<tr>
<td>9</td>
<td>Select (same as ENTER)</td>
</tr>
<tr>
<td>A</td>
<td>Updir</td>
</tr>
<tr>
<td>B</td>
<td>Downdir</td>
</tr>
<tr>
<td>C</td>
<td>Previous menu</td>
</tr>
<tr>
<td>D</td>
<td>Next menu</td>
</tr>
<tr>
<td>E</td>
<td>EVAL</td>
</tr>
<tr>
<td>F</td>
<td>Swap header</td>
</tr>
<tr>
<td>18</td>
<td>Display or remove file's details (Type &amp; size)</td>
</tr>
<tr>
<td>19</td>
<td>EDIT</td>
</tr>
<tr>
<td>1A</td>
<td>COPY</td>
</tr>
<tr>
<td>1B</td>
<td>MOVE</td>
</tr>
<tr>
<td>1C</td>
<td>RCL</td>
</tr>
<tr>
<td>1D</td>
<td>PURGE</td>
</tr>
<tr>
<td>1E</td>
<td>RENAME</td>
</tr>
<tr>
<td>1F</td>
<td>CRDIR</td>
</tr>
<tr>
<td>20</td>
<td>ORDER</td>
</tr>
<tr>
<td>21</td>
<td>SEND</td>
</tr>
<tr>
<td>22</td>
<td>HALT2</td>
</tr>
<tr>
<td>23</td>
<td>EDITB</td>
</tr>
</tbody>
</table>

e) Custom program

If you are going to run a custom program there are 7 ways to call the program. Every time, the filer will place the working path on the stack.

Examples:

```
<>                     for HOME
:e:<>:<>               for a port
:02: (<FOO.DIR>)      if you are working in port 2 in the backup FOO.DIR
```

If you launch the program in the VAR, then your program will start in the current directory. The possible calls are: (1: ect is the resulting stack that will appear just before your program is run)

<table>
<thead>
<tr>
<th>SB</th>
<th>Type of call</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Recalls only the path:</td>
</tr>
<tr>
<td></td>
<td>1: Path</td>
</tr>
<tr>
<td>11</td>
<td>Recalls the name and the object for the currently selected object:</td>
</tr>
<tr>
<td></td>
<td>3: Path</td>
</tr>
<tr>
<td></td>
<td>2: Object</td>
</tr>
<tr>
<td></td>
<td>1: Name</td>
</tr>
<tr>
<td>12</td>
<td>Multiple objects selected</td>
</tr>
<tr>
<td></td>
<td>n: Path</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>5: Object 2</td>
</tr>
<tr>
<td></td>
<td>4: Name 2</td>
</tr>
<tr>
<td></td>
<td>3: Object 1</td>
</tr>
<tr>
<td></td>
<td>2: Name 1</td>
</tr>
<tr>
<td></td>
<td>1: Number of objects (System Binary)</td>
</tr>
<tr>
<td>13</td>
<td>The FILER will call your program after each object. The FILER will place the following on the stack each time and call your program for each object selected.</td>
</tr>
<tr>
<td></td>
<td>3: Path</td>
</tr>
<tr>
<td></td>
<td>2: Object 1</td>
</tr>
<tr>
<td></td>
<td>1: Name</td>
</tr>
<tr>
<td>14</td>
<td>Recalls the name of the current object only</td>
</tr>
<tr>
<td></td>
<td>2: Path</td>
</tr>
<tr>
<td></td>
<td>1: Name</td>
</tr>
<tr>
<td>15</td>
<td>Recalls all the names selected</td>
</tr>
<tr>
<td></td>
<td>n: path</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>3: Name2</td>
</tr>
<tr>
<td></td>
<td>2: Name1</td>
</tr>
<tr>
<td></td>
<td>1: Number of names selected (System Binary)</td>
</tr>
<tr>
<td>16</td>
<td>Recalls the current object only in a string of addresses</td>
</tr>
<tr>
<td></td>
<td>2: Path</td>
</tr>
<tr>
<td></td>
<td>1: String</td>
</tr>
</tbody>
</table>
Recalls the selected objects in a string of addresses

2: Path
1: String

f) Explanation for the string of addresses
This string recalled by the filer is a list of addresses. You can use it only in ML or with FILER_FRCL, FILER_FNAME and FILER_NXTADR, for SysRPL use:
FILER_FRCL (C030F): Takes a String of addresses, and recalls the first object. FILER_FNAME (C030A): Same thing but recalls the first name object, FILER_NXTADR (C0314): Takes a string of addresses, and removes the first addresses. And FALSE if there are some addresses left. TRUE if not.
The string format is:
Prolog (02A2C) Size AdrName1 AdrObject1 AdrName2 AdrObject2 ...
Each address is on 5 nibbles.

g) Explanations about the name
For all calls except 14 and 15, the name is the real filename. If you call the program from a library, then the name will be the name of the library itself (Like 'Str.wrt: v4.31 (c)'). However, for 14 and 15, if you have selected a library, the name will be a real number that is the library number. For 13, if the object is a library, then the name will be a 'L' plus the Id number.
Example: For Library 1303, the name will be: 'L1303'

NOTE: You can browse every library, but can only run your programs from an attached library.

h) Warning and very important points
In order to save room, we used list of addresses. So, if you modify the directory structural in your programs (using STO, PURGE, etc.), then NEVER USE CALL 13. And if you are a very experienced programmer, you can use calls 16 and 17.

4.2 Machine Language entry points list
Here is a list of entry points inside the Meta Kernel. They can be used by experienced programmers for programs to take advantage of the MK. Of course, if these entry points are used, the program won't run without the MK.

Call a routine with G0SBVL C9.XXX.
Rf means the R register with the field f. Example: Ca describes the register C, field A.
Input and output registers are described in the ‘In’ and ‘out’ sections.

4.2.1 MINI_DISP
Address 0040
Description Displays a Mini-font character string
In
Ca = number of characters
D1 = string address
D0 = display starting address (in a 131*x grob)
ST11 = inversion (white on black)

4.2.2 MINI_FONT_ADDRESS
Address 0047
Description Mini-font address in the card
E.g. D0=C0047 R=DRT0.R

4.2.3 DISPLAY_SBR
Address 004C
Description Main display routine
Call INIT_DISPLAY_LINE before using this routine.
In
R0b = font identifier
Rs = (font height) * 2 - 1
D0 = starting display address in a grob
D1 = string address
Da = number of characters
R2a = display grob width in nibbles
Ba = left margin in characters
Ca = right margin in characters
ST4 = display left shift
ST5 = display right shift
ST0,1,2,3 = text attributes (bold, italic, underline, inverse)
ST8 = display carriage returns as characters if set
ST9 = mini-font display
Out
D0 = next line address in the grob
Aa = last written character address
Da = number of remaining characters
ST8 = set if a CR has been read
D1 = end of displayed text in the grob

4.2.4 EDIT_SBR
Address 0053
Description Mini-edit (used by FIND in the command line)
Call INIT_DISPLAY_LINE before using this routine.
In
R4a = screen address
Ca = maximum number of characters
D1 = buffer address
ST9 = mini-font display
Out
Carry = set if ENTER has been pressed, clear exit by DN

4.2.5 FILER.ADR
Address C0319
Description Address of the main FILER program

4.2.6 SCAN_FONT
Address C009R
Description Rebuild the font table
This function forces the MK to search through all ports to build the font table.

4.2.7 RECONFIG
Address C00R1
Description Reconfigures the second card
After a DECONFIG, this routine reconfigures the second port card at C0000 (hidden by the MK card), and the bank switcher at 7F000.

4.2.8 DECONFIG
Address C00R8
Description Deconfigures the second card
This routine is used to gain direct access to the second card, without deconfiguring the MK.
The second port card is moved from C0000 (where it is hidden by the MK card) to 40000. The bank switcher is moved from 7F000 to 3F000.

4.2.9 CMD_SIZE
Address C00BE
Description Computes the command line size
Out
Aa = number of characters.

4.2.10 GET_PATH
Address C00C5
Description Finds the relative path of a directory
In
D1 = buffer address
Ca = maximum length
If ST2 = 0, the path is taken from the HOME directory to the current directory
If ST2 = 1, D0 = starting directory address, Da = ending directory address

4.2.11 DECONFIG_RAM
Address C00CC
Description Moves the RAM from 80000 to 00000
Useful if you plan to make your own interrupt handler. If your RAM based program call this entry point, the return address (RTN) will be recalculated

4.2.12 RECONFIG_RAM
Address C00D3
Description Moves the RAM from 00000 back to 80000

4.2.13 GET_FONT
Address C00DR
Description Returns the address of a given font
Call SCAN_FONT once before using this routine.
In
Ab = font id
Out
R0a = font address

4.2.14 INIT_DISPLAY_LINE
Address C00E1
Description Initializes the display routines
This routine initializes the registers necessary for the MK display routines.
In
Ca = screen width in nibbles

4.2.15 CHANGE_FLAG
Address C00E8
Description Updates the ST flags 0 to 3 and R0 for the display routine
Computes some parameters for the display routines, when special characters are encountered (font control characters...)
4.2.16 GET_ASCII_KEY
Address C00F6
Description Keyboard handler
Gets an ASCII character or a key number if not in ALPHA mode. This routine handles the shifts and alpha displays.
Out
ST0 = 0 if ASCII
ST0 = 1 if key number.

4.2.17 GET_KEY
Address C00FD
Description Keyboard handler
Out
Carry = set if no character in the keyboard buffer.
Aa = key code (if carry cleared)

4.2.18 RUN_KEY
Address C0104
Description Key assignment
This routine associates an assembly code with a key.
Syntax:
GOSBVL =RUN_KEY
GOIN4 NO_KEY_ROUTINE
# /01 GOIN4 KEY_R % 01 is the A key code
# /xG GOIN4 KEY_x % xx key code (same as in External)
...
# $0 GOIN4 NOT_HRNDLED_KEY % a bip routine for example

4.2.19 MINI_DISP_VAL
Address C010B
Description Displays a hexa value in mini-font
In
D0 = screen address
Bw = number to be displayed
Ca = number of digits to be displayed
ST11 = inverse (white on black)
ST10 = 1 to display leading zeros

4.2.20 MINI_DISP_AWP
Address C0112
Description Displays a hexadecimal value in mini-font
In
D0 = screen address
Awp = number to be displayed
P = (number of digits) - 1
ST11 = inverse (white on black)
ST10 = 1 to display leading zeros

4.2.21 TRUP
Address C0119
Description Copy up
Equivalent to $66B9, but works even after a DECONFIG_RAM (when RAM is at $0000)

4.2.22 TRDN
Address C0120
Description Copy down
Equivalent to $678C, but works even after a DECONFIG_RAM (when RAM is at $0000)

4.2.23 ZEROM
Address C0127
Description Resets a memory zone to 0
In
D1 = zone starting address
Ca = number of nibbles to blank

4.2.24 SCAN_KEY
Address C012E
Description Keyboard handler
Does a keyboard scan, updates the keyboard buffer and the keystate. Use it when interrupts are disabled.

4.2.25 NEW_ADR
Address C0135
Description Moves an object at the end of the temporary zone
Moves an object in the temporary RAM at the end of this zone, so that $6671 (object resize) and RESIZE_PLUS can be used on it.

4.2.26 RESIZE_PLUS
Address C013C
Description Resize up a character string
Resize up the last character string in the temporary RAM.

In
\[ R0a = \text{string address} \]
\[ D0 = \text{new end of the character string} \]

4.2.27 C=IN3

Address \( \text{C02F2} \)
Description Equivalent to C=IN2, but in the MK card

4.2.28 A=IN3

Address \( \text{C02FA} \)
Description Equivalent to A=IN2, but in the MK card

4.2.29 OUT=C=IN3

Address \( \text{C02EF} \)
Description Equivalent to OUT=C=IN, but in the MK card

4.2.30 OUT=CA=IN3

Address \( \text{C02F7} \)
Description Equivalent OUT=C A=IN, but in the MK card

4.2.31 DISP_DEC

Address \( \text{C0143} \)
Description Displays a decimal number

In
\[ D0 = \text{screen address} \]
\[ Ca = \text{number to be displayed} \]
\[ ST11 = \text{reverse (white on black)} \]
\[ ST10 = 1 \text{ to display leading zeros} \]

4.2.32 MULT_BAC

Address \( \text{C014A} \)
Description \( Ba = Ca \times Aa \)

4.2.33 GREY?

Address \( \text{C0151} \)
Description Tests if a grob is in grayscale

In
\[ D1 = \text{grob address} \]
Out
\[ \text{Carry} = \text{set if grob is in grayscale} \]

4.2.34 BEEP

Address \( \text{C0158} \)
Description Do a beep if the system flag –56 is clear

4.2.35 KEY_REPEAT

Address \( \text{C015C} \)
Description Keyboard handler routine
Repeats a routine associated with a key.
Syntax:
\[ \text{GOSBYL =KEY\_REPERT} \]
\[ \#\langle\text{key mask}\rangle \% \text{in/out mask}, \text{e.g. } \$\langle02040 \text{ for OUT=040, IN=02} \)
\[ \text{GOTO \_KEY\_REPERT_ROUTINE} \]
\[ \text{GOTO END\_KEY\_REPERT} \]

4.2.36 KEY_NO

Address \( \text{C0160} \)
Description Waits for all keys to be released

4.2.37 HEX_DEC

Address \( \text{C0164} \)
Description Hexa to decimal conversion

In
\[ Ca = \text{hexa to be converted} \]
Out
\[ Aa = \text{decimal result} \]

4.2.38 A\_DIV\_C

Address \( \text{C0168} \)
Description \( Ba = Aa / Ca \)

4.2.39 SET_BIT

Address \( \text{C016C} \)
Description Sets a bit in a bit field

In
\[ D1 = \text{bit field address} \]
\[ Aa = \text{bit number (to be set to 1)} \]
4.2.40 CLEAR_BIT
Address    FEC3F
Description Clears a bit in a bit field
In
D1 = bit field address
Aa = bit number (to be clear)

4.2.41 BIT?
Address    C0179
Description Tests a bit in a bit field
In
D1 = bit field address
Aa = bit number (to be tested)
Out
Carry = set to the value of the tested bit

4.2.42 BZU.SBR
Address    C017A
Description BZU (BZ Uncompression)
Uncompress data using Mika Heiskanen algorithm in his program BZ.
In
D0 = Address of the object to decompress (strings without the B prolog BZ).
D1 = Address where you want to uncompress.

4.2.43 INV.ZONE
Address    C0186
Description Invert a memory zone
In
D0 = Address zone.
Ca = Size in nibbles to inverse

4.2.44 OFF.SBR
Address    C0192
Description Turns off the machine (real Off)
OFF in ML, doesn’t reboot if libraries have changed.

4.2.45 InitTable
Address    FEC00
Description Initializes memory zones for SysRPL-table routines
 Initializes pointers used by the following routines, which deal with the System RPL entry point table.
 There are examples on the floppy disk, in the file EXAMPLES2.DIR.
 Warning: Port 2 must be at address 40000. Use the DECONFIG and RECONFIG routines.
Out
Carry set if no table has been found.
Uses RSTK(5) Aw Cw Ba D0 D1 Da, plus the following memory zones:
(b=bytes, n=nibbles)
DC@ResZone BE4AR % on 256b, table name
DC@Exports BE4R2 % on 5n, number of entries
DC@TableText BE427 % on 5n, address of the first entry
DC@TableHash BE42C % on 5n, address of the hash table
DC@Table+0 BE431 % on 5n, address of the pointers table, sorted by addresses
DC@TablePort BE436 % on 2n, port number of the table
DCRc1Port BE31F % on 2n, port number of the table

4.2.46 GetAdr
Address    FEC0C
Description Returns the address associated with a name
Warning: Port 2 must be at address 40000. Use the DECONFIG and RECONFIG routines.
In
DO = Address of the text to find, followed by a separation character
Out
Carry = Set if name not found
Ca = Address
Uses Aw, Cw, Ba, Da, D0, D1, RSTK(5).
DC@ResZone BE4AR % on 256b, name saved

4.2.47 GetText
Address    FEC06
Description Returns a pointer to the name associated to an address
Warning: Port 2 must be at address 40000. Use the DECONFIG and RECONFIG routines.
In
Ca = Address
Out
Carry = Set if address not found
D1 = Address of the text (2n for the length, followed by the characters)
Uses Aa, Ba, Ca, Da, D0, D1, RSTK(2).
4.2.48 GetFirst
Address FECB2
Description Finds the first name beginning with the given text
Warning: Port 2 must be at address 40000. Use the DECONFIG and RECONFIG routines.
In
  DO = Address of the text to find, followed by a separation character
Out
  Carry = Set if no name found
  D1 = Address of the text (2n for the length, followed by the characters)
Uses Aw, Ba, Cw, RSTK(4), D0, D1, Da
DC@ResZone BE4R5 % saving of the name to find (on 256 bytes)

4.2.49 GetNext
Address FECB8
Description Finds the next name beginning with the given text
Use GetFirst first, values of Bb, Da and D1 must be preserved.
Warning: Port 2 must be at address 40000. Use the DECONFIG and RECONFIG routines.
In
  DO = Address of the text to find, followed by a separation character
Out
  Carry = Set if no name found
  D1 = Address of the text (2n for the length, followed by the characters)
Uses Aw, Ba, Cw, RSTK4, D0, D1, Da
DC@ResZone BE4R5 % saving of the name to find (on 256 bytes)

4.2.50 GetFirstAdr
Address FECBE
Description Returns a pointer to the first table entry
Warning: Port 2 must be at address 40000. Use the DECONFIG and RECONFIG routines.
Out
  D0 = Address of an entry structure:
    Address (5n), name length (2n), name.
Uses D0, D1, Aa, Da, Ca, RSTK(2).

4.2.51 GetNextAdr
Address FECC4
Description Returns a pointer to the next table entry
Use GetFirstAdr first, values of Da and D1 must be preserved.
The table is sorted by addresses.
Warning: Port 2 must be at address 40000. Use the DECONFIG and RECONFIG routines.
Out
  Carry = Set if no more entries
  D0 = Address of an entry structure (see GetFirstAdr)
Uses D0, D1, Aa, Da, Ca, RSTK(2).

4.2.52 RclPath
Address FEC39
Description Finds the address of a file
Warning: Port 2 must be at address 40000. Use the DECONFIG and RECONFIG routines.
In
  DC@ResZone BE4R5 % on 256b, filename (Masd syntax, ends with an ASCII 0)
Out
  Carry = Set if no file found
  D1 = Address of the file
Uses Aw, Ba, Cw, RSTK4, D0, D1

4.3 SysRPL entry points
Here is a list of entry points inside the Meta Kernel. They can be used by programmers for programs to take advantage of the MK.
Of course, if these entry points are used, the program won't run without the MK.
Call a routine with /c51/c55/c53/c3/c91/c91/c91 in MASD, or with /c111/c3/c91/c91/c91.

4.3.1 GET.INDEX
Address FEC96
Description Create an index for a fast access
In
  1 : Matrix (List of list)
Out
  2 : Matrix (list of list)
  1 : Index (Strings)

4.3.2 GET .Y.I
Address C006A
Description Recall a row from a matrix
In
  3 : Matrix (List of list)
  2 : Index (String)
### 4.3.3 GET .X.Y.I

**Address**  \( \text{C0305} \)

**Description**  Recall a cell

**In**
- 4: Matrix (List of list)
- 3: Index (String)
- 2: X (System Binary)
- 1: Y (System Binary)

**Out**
- 3: Matrix (List of list)
- 2: Index (String)
- 1: Object

### 4.3.4 PUT.X.Y

**Address**  \( \text{FEC64} \)

**Description**  Put an object into a cell

The matrix has to be split using INNERCOMP (\$ 54AF) ex:

\[
\{ \{ 1 2 3 \} \{ 4 5 6 \} \} \text{ INNERCOMP :}
\]

\[
3: \{ 1 2 3 \}
2: \{ 4 5 6 \}
1: <2>
\]

**In**
- N: First row
- ...
- 5: Row n
- 4: Number of rows (System Binary)
- 3: X (System Binary)
- 2: Y (System Binary)
- 1: Object

**Out**
- N: First row
- ...
- 2: Row n
- 1: Number of rows (System Binary)

### 4.3.5 DIMS

**Address**  \( \text{FEC9B} \)

**Description**  Give the dimensions of a matrix

**In**
- 1: Matrix (list of lists)

**Out**
- If it's a matrix
  - 4: Matrix
  - 3: X (System Binary)
  - 2: Y (System Binary)
  - 1: TRUE
- or
  - 1: FALSE

### 4.3.6 FILER_FNAME

**Address**  \( \text{C030R} \)

**Description**  See page 52

### 4.3.7 FILER_FRCL

**Address**  \( \text{C030F} \)

**Description**  See page 52

### 4.3.8 FILER_NEXTADR

**Address**  \( \text{C0314} \)

**Description**  See page 52

### 4.3.9 SURPRISE

**Address**  \( \text{C031E} \)

**Description**  Just try!

### 4.3.10 INPUT

**Address**  \( \text{C0800} \)

**Description**  SysRPL INPUT, works like InputLine (\( \text{42F44} \))

This INPUT command takes 10 arguments. [more in complete documentation].
4.3.11 CTRL_LOOP
Address C0085
Description Parameterized outer loop, like ParOuterLoop (38985).
Takes 9 arguments. [more in complete documentation].

4.3.12 INIT_CMD
Address C0080
Description Initializes the command line

4.3.13 CMP_PLUS
Address C0090
Description Adds a character to the command line string

4.3.14 DISP_CALL
Address C0095
Description Text display
In
2: Character string
1: TRUE if carriage returns are to be interpreted

4.3.15 BZU
Address C018D
Description Uncompress with BZU
Uncompress the compressed strings using Mika Heiskanen’s algorithm from his BZ program.
In
1: Strings (without its BZ prolog).
Out
1: Object

4.3.16 KER_PARAM
Address C0181
Description OuterLoop
Internal version of the ParOuterLoop. This program doesn’t create a local environment to save the menus, therefore you can use unnamed local variables.

4.3.17 MINI_EDITOR
Address C0199
Description Mini editor on one screen line
In
3: String (if boolean on level 2 is TRUE)
2: Boolean
1: Line number, where the command line has to be put.
Out
2: String
1: Boolean (TRUE if line validated by ENTER, else FALSE)
If the boolean is TRUE, the editor is initialised with the character string on level 2.

4.3.18 KEY_PARAM
Address FEC82
Description Recall a program associated to a key
Assigns a program to key. Recalls the corresponding program.
In
3: Key code
2: Shift code
1: program list (string)
The list is as follow in MASD format:
STRING (  
%\$a\$xx e \$xx\$xx 
%: a is 1 if in alpha mode, else 0, xx is the code of the key, plus the ML shift code; %: RightShift=C0 ; LeftShift=40. For example for the LeftShift+ENTER, run the %internal command DUP :
%$9B59 EXP<5>DUP 
%$00 %:at the end 
) 
Out
If the key is in the list:
2: program
1: TRUE
Otherwise
1: FALSE

4.3.19 UPSTACK.ADR
Address FEC69
Description Internal programs list
Recalls the programs list used by the Interactive Stack. See KEY_PARAM.
Out
1: String

4.3.20 WAITKEY.ADR
Address FEC6E
Description: Internal programs list
Recalls the programs list used by the Kernel. See KEY_PARAM.

Out
1: String

4.3.21 COMMANLINE.ADR

Address: FEC73
Description: Internal programs list
Recalls the programs list used by the command line editor. See KEY_PARAM.

Out
1: String

4.3.22 MATRIX.ADR

Address: FEC87
Description: Internal programs list
Recalls the programs list used by the Matrix Writer. See KEY_PARAM.

Out
1: String

4.3.23 SAVE.ARG

Address: FEC78
Description: Save the LASTARG
This program is very useful, while you want to run a User RPL command, and you don't want this program to save the last arg. See LOAD.ARG too
Ex:
"1 NO CODE !RPL E0U SAVE.ARG FEC78 E0U LOAD.ARG FEC7D :: SAVE.ARG xDUP LOAD.ARG ; e" In case of errors, the program will restore the good stack.

4.3.24 LOAD.ARG

Address: FEC7D
Description: Restore the LASTARG saved by SAVE.ARG
See above
5 MetaKernel 3.0 : Preview

Here are some screenshotted. Just pour le plaisir des yeux

Pretty Print on numerical matrices

Symbolic matrices

Symbolic Expressions

Units in pretty print

Erable results in pretty print

View big objects directly on the stack

Multi-Tasking: Switch between applications on the fly

Algebraic Mode

And...

Equation Writer 3: Edit symbolic matrices, unit objects
INFORM2: Speed up every menus of the HP48GX by up to 500 times
Modular: Select exactly what you want to use in the MetaKernel
And MUCH MORE ....

Coming Soon...