HP48 Hacker's ROM

OWNER'S MANUAL

HP48 Hacker's ROM

July 26, 1995

OWNER'S MANUAL

CONTENTS

<-LIB-> library, by Detlef Mueller 3
Converts libraries to directories and vice versa, and more.

Jazz library, by Mika Heiskanen 14
Complete software development tool, including assembler, disassembler, and debugger for System RPL and Assembly Language; Entry Point Catalog with Search and Disassembly; and a String Editor with special features for source code editing.

Library 993, the Jazz Entry Point Table Library 26
Makes Jazz run very fast (assemble/disassemble hundreds of lines per second!).

LIBEX (Library Explorer) by Marc Vogel & Régis Duchesne 31
Easiest way to make a subset of a library. It extracts one or more commands from any library and automatically extracts all of the referenced library calls as well. The result is placed in a directory so that <-LIB-> can turn it into a library.

STR33 by Todd Eckrich 32
Formats any length string to 33-char width lines, with word wrap.

ED33, by Joseph K. Horn 32
Replacement for the built-in editor. Uses Jazz's ED.

HACK Library, by Mika Heiskanen 33
Replacement for Rick Grevelle's HACKIT library, plus lots more.

SPORT (Search PORT) by Dave Marsh and Joseph K. Horn 36
Searches GX ports 2+ for any name or library ID.

VV33 by Joseph K. Horn 36
Views any object. Uses Jazz's VV.

MKCOPY by Rick Grevelle and Joseph K. Horn 37
Sets up HP48 for copying ROM cards to multi-port RAM cards.

Code, by Joseph K. Horn and Richard Steventon 38
Converts any object to a Code object and back again.

FBROW by Richard Steventon 39
Fast flag browser with Jazz flags (and many others) identified.

UPD by Joseph K. Horn 40
Improved version of UPDIR, especially useful for programmers.

BZM (BZ Menu) by Jack Levy 40
Creates a menu for easy use of the BZ object packer/unpacker.

HTRIM (Home-directory Trimmer) by Simone Rapisarda 41
Makes selected vars in HOME fully accessible but totally invisible!

About.HackCard -- brief info about the HP48 Hacker's ROM.

WHAT IT IS

The HP48 Hacker's ROM is a 128K collection of the best software available for System RPL and Assembly Language program development and maintenance for the HP48 SX and GX, most notably the Jazz library by Mika Heiskanen. The selection and compilation of the software and its documentation was made by Joseph K. Horn. The ROMs were produced by EduCALC as a service to the HP48 enthusiast; EduCALC sells the HP48 Hacker's ROM at cost, and is making no profit from them.

This documentation assumes the reader is familiar with the terminology and methodology of HP48 hacking. A great primer for beginners is Jim Donnelly's book, "An Introduction to HP 48 System RPL and Assembly Language Programming" available through EduCALC.

WHY IT IS A GOOD IDEA

Jazz is unquestionably the best hacking tool. But it is over 95K bytes in size, so keeping it in a 32K RAM card is impossible. ROM's (OTP's) are cheaper than RAM, so it's best to burn Jazz into a 128K ROM. That leaves 33K bytes available for other hacker's goodies, which we filled with a baker's dozen of libraries and programs specifically designed for hackers. So you can develop software with this HP48 Hacker's ROM, and use your expensive RAM card(s) more wisely.

COPYRIGHT NOTICE

The libraries and programs in the HP48 Hacker's ROM are by different authors; some are freeware, and some are giftware. See the copyright notice for each in its documentation below. The collection as a whole may be copied if and only if this entire document is also copied. The person providing the copy of the ROM and this documentation may recoup costs only, and may not profit from its sale.

DISCLAIMER

The HP48 Hacker's ROM carries no warranty of any kind. It is offered on an "as-is" basis only. It is NOT an HP or EduCALC product, and is not guaranteed or supported in any way by HP, EduCALC, Joseph K. Horn, nor any of the authors of the software contained in the card. It uses (and makes it possible for the user to access) low-level functionality of the HP48, and as such it can corrupt memory contents, clear memory, and even cause damage to the HP48's hardware. THESE ARE THE DANGERS THAT ALL HACKERS FACE ON A REGULAR BASIS. If you cannot accept these dangers, then do not use this card. Use of this card constitutes an agreement to personally accept the consequences of hacking.

Copyrights & Acknowledgements

We would like to thank the following people for their support:

Wlodek Mier-Jedrzejowicz for presenting <-LIB-> to the world.

Rick Grevelle	for the HACKIT library, the ->DIR command of <-LIB->, the base of the sys-stack, many suggestions, exiting talks and for sending us an HP48GX.
Mika Heiskanen	for beta testing, lots of suggestions, MKROM, DEBUG, and for a great performance enhancement of the D->LIB and L->DIR commands.
Douglas R. Cannon	for beta testing, lots of suggestions, and for reviewing this document.
Joseph K. Horn	for beta testing and suggestions, the 'HP 48 Resource Allocation Guideline; Library ID's', SORTLS and for maintaining the '48 GDs.
Carlos Ferraro	for beta testing and his friendship.
Simone Rapisarda	for beta testing and many suggestions.
Fatri Mohamed, Romain Desplats, Georg Hoppen	for suggestions and beta testing.
James H. Cloos, Steve VanDevender, Chris Maksymiak	for suggestions.
Jeoff Krontz,	for suggestions, beta testing, getting another GX for Raymond, exiting talks, and..
Chris Spell	..for maintaining the '48 archive at seq.uncwil.edu and for moderating comp.sources.hp48.
Dennis York	for the kindly permission to publish the BNF parser generator example.

W.C.Wickes & HP Corvallis for the '48 and the RPL tools.

Abbreviations

LID - library id; a number in the range 0..2047, part of any library, It is used by the HP firmware to identify libraries while resolving commands, messages etc.
| - used in stack diagrams for 'or'

Abbreviations denoting objects:

```
ob      object of any type
xlib    named or unnamed library command
:x:y    tagged object - tag x, ob y
prg     program (secondary)
bak     backup
dir     directory
alg     algebraic
lib     library
libdta  library data
meta(t) meta object, obn .. ob1 %n, all ob's are of the same type t
```

All other abbreviations are defined in RPLMAN.DOC (like %, # etc.).
[Available on GD4. -jkh-]

Overview

<-LIB-> contains 39 commands, including a library maker (D->LIB), a library splitter (L->DIR), commands to handle any sort of composite objects and commands to manage libraries.

Libraries are very useful objects to extend the command set of a '48 in a 'native' manner, but unfortunately library creation is not supported by the '48 firmware. The HP tools package contains the program USRLIB.EXE, which provides library creation on a PC; if you want to move your most often used programs into a library (because of easier handling and to keep your VARs area clean), you have to collect them into a subdirectory, add a few control variables, transfer the directory in binary form to a PC, run USRLIB.EXE on it, and transfer the created library back to your '48, where you can install and test the library.

<-LIB-> provides the command D->LIB that works on the currently active directory, allowing you to create a library from this directory on the '48 itself - no transfers, no PC, and no USRLIB.EXE are required.

The input directory stucture of D->LIB is very similar to the directory structure required by USRLIB.EXE, thus existing USRLIB.EXE input can be used for a library creation with minor changes (the difference is the format of the $MESSAGE control variable, see below).

Control Variables

The library creation process is controlled by some variables with reserved names and only the current directory is searched for their presence (note: multiple occurences of these names are ignored, only the contents of the first one found in the current directory is used for the library creation process):

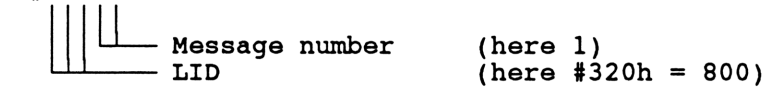
$ROMID
: Must contain a real or binary number representing the LID that is to be given to the library. The LID must be in the range 0..2047. Negative real numbers are mapped to 0. This is the only control variable that MUST exist in the current directory !

$TITLE
: Should contain a character string to be used as the name of the library (note: any other object is converted to a string). The first few characters of the name are displayed in the LIBRARY menu label associated to the library, the first 22 characters are displayed by pressing REVIEW in the LIBRARY menu. If $TITLE is absent or contains an empty string, no title is generated and the resulting library doesn't have a menu label in the LIBRARY menu (note: you can switch to the library's command menu by executing %LID MENU, even if the library doesn't have a name or is not

attached).

$CONFIG
Must contain a program to be executed at configuration time. The configuration code can generally NOT be written in User-RPL, but simple programs such as \<< 123 ATTACH \>> are Ok - more complicated programs should take care to leave the stack unchanged, and be sure NOT TO ERROR ! An error in a configuration program will cause a warmstart; the configuration code is called again during the startup process, producing a new error etc. If you are accidentally trapped by this, you must clear your memory to remove the faulty library !

$MESSAGE
Must contain a list of strings to be combined into a message table (note: any other objects are converted to strings). The message numbers correspond to the list positions. In User-RPL you can generate errors with your own messages in the following manner:



In Sys-RPL you can use
32001 ERROROUT

In this example an error is generated, using the first message from the message table of a library with the LID 800.

Note: The message list structure is NOT compatible to USRLIB.EXE.

$VISIBLE
Must contain a list of names of variables to be converted to user-accessible, named library commands. By default, all variables will be translated to named library commands. When the $VISIBLE list is present, only the names in this list are included in the library hash table. An empty list is Ok, meaning that no hash table is generated.

$HIDDEN
Must contain a list of names of variables that are to be converted to unnamed objects in the library. When the $HIDDEN list is present, those names listed are not entered in the library hash table. If both $VISIBLE and $HIDDEN are present, only $HIDDEN will be used.

Note: all visible commands appear as labeled softkeys in the menu associated to the library, all hidden commands are not, and there is no way to access them from user scope.

$VARS
Must contain a list of variables that should remain RAM-based - i.e. the objects of the variables listed in $VARS are not included in the library and no XLIB pointers are made to substitute their names. All other variables of the current directory are included in the library.

Note: You should add all subdirectory names of the current directory to the list.

$ROMID must exist in the current directory, all other control variables are optional - eg. you can generate libraries containing only a configuration program or a message table. All control variables are internally handled by D->LIB to be $VARS and can't be accessed from within the library commands.

D->LIB (--> lib)

Dir to lib; assembles a library from the objects of the current directory. The creation process is controlled by a few variables with reserved names (see '3.3 Control Variables').

If you set flag -13 before running D->LIB, it will switch off the screen while working (for saving time and batteries).

L->DIR (%LID --> dir)
(xlib --> ob)

Lib to dir; if the argument is a LID then L->DIR assembles a directory containing all commands of this library as variables. The neccessary control variables (see 3.3) are also generated. D->LIB may be used to recreate the library.

If the argument is an XLIB, then L->DIR recalls its object from the associated library onto the stack.

If you set flag -13 before running L->DIR on a library, it will switch off the screen while working.

Note: You cannot use L->DIR to split <-LIB->, this should prevent users from a memory lost. If you are familiar with the internals of your '48, use 'ROMPTR@' (or XRCL in HACK) to extract single routines from <-LIB->, but beware of starting the routines in RAM, most of them will crash your calc when running alone !

MCFG (-->)

Make config; stores a configuration program into a variable named '$CONFIG' into the current directory. This configuration program will attach the generated library to the home directory at warmstarts (ON-C etc.).

'$ROMID' must exist in the current directory.

ML->D (--> prg)

Generates a program with the following interface:

({} --> libdta)

The list can contain anything. The program checks for argument count and type and may error with:

#201 - To Few Arguments - nothing on the stack
#202 - Bad Argument Type - not a list on the stack

'$ROMID' must exist in the current directory.

MD->L (--> prg)

Generates a program with the following interface:

(libdta --> {})

The program checks for argument count, type and correct LID and may error with:

```
        #201 - To Few Arguments   - nothing on the stack
        #202 - Bad Argument Type  - no library data on the stack
        #203 - Bad Argument Value - the libdta wasn't created by this
               lib
```

'$ROMID' must exist in the current directory.

Note: Store the programs generated by ML->D and MD->L into variables in your source directory and use them as an interface to generate/resolve data associated to the resulting library.

```
OB->            ( prg                   --> ob1 .. obn %n              )
                ( xlib                  --> %LID %objno                )
                ( arry                  --> ob1 .. obn { %di .. %d1 }  )
                ( alg                   --> ob1 .. obn %n              )
                ( dir                   --> ob1 id .. obn id %n        )
                ( id                    --> $                          )
                ( libdta                --> {} %LID                    )
                ( bak                   --> ob id                      )
                ( id                    --> $                          )
                ( ob                    --> dispatch to OBJ->          )
```

OB-> is an extension to the built-in OBJ->, supporting system level objects.

```
->DIR           ( ob1 id1 .. obn idn %n --> dir                        )
->PRG           ( ob1 .. obn %n         --> prg                        )
->XLIB          ( %LID %cmdno           --> xlib                       )
                ( #LID #cmdno           --> xlib                       )
->ARR           ( ob1 .. obn %n         --> arry                       )
                ( ob1 .. obn { %di .. %d1 } --> arry                   )
->ALG           ( ob1 .. obn %n         --> prg                        )
->LD            ( {} %LID               --> libdta                     )
->BAK           ( ob id                 --> bak                        )
->ID            ( $                     --> id                         )
```

Functions to reverse OB->.

About ->ARR:
Generates arrays of any type and any dimension (eg. a four dimensional array of libraries :-).

%di * .. * %d1 must be = n, ob1 .. obn must be of the same type. All possible parameter errors are trapped.

```
ADRp            ( ob                    --> ob #addr                   )
```

Get address of ob.

Note: The data stack is a stack of pointers to objects. ADRp simply returns the value from the top element (about the 'p', see 'LIBp' below).

```
$romid
$visible
$title
$config
$vars
$hidden
$message        (                       --> '$XXX'                     )
```

These commands just put a control variable name onto the stack.

```
LBCRC           ( lib                   --> lib'                )
                ( bak                   --> bak'                )
```

Recalculates the CRC of a library or backup, useful if you have patched it because modifying the body of a library or backup invalidates the CRC included at the end of the body.

```
RNLIB           ( lib $                 --> lib'                )
```

Renames a library, ie. changes title.

```
CHLID           ( lib %                 --> lib'                )
```

Change LID; this program allows you to change the LID of a library if it's not splittable.

Note: Most time a LID is also hardcoded in the config code - this can generally not be changed by CHLID. You have to attach the library manually after a warmstart. Also any library has its LID coded in a field above any visible command (the error handling system identifies the command that caused an error using this field (the Sys-RPL commands 'CKn' copies this values to the appropriate location)). These fields are not changed by CHLID.

```
RHASH           ( %LID                  --> hxs                 )
```

Recall hash table; get a pointer to the hash table of a library.

```
RLINK           ( %LID                  --> hxs                 )
```

Recall link table; get a pointer to the link table of a library.

```
RCFG            ( %LID                  --> ob                  )
```

Recall config code; get a pointer to the config code of a library.

```
RMSG            ( %LID                  --> arry                )
```

Recall message table; get a pointer to the message table of a library.

```
RTITLE          ( %LID                  --> $                   )
```

Recall title; get the name of a library.

Note: RHASH, RLINK, RCFG, RMSG and RTITLE don't error if the library associated to %LID didn't contain the requested item, they leave the stack unchanged in that case.

```
RPORT           ( %port                 --> ob1 .. obn          )
```

Recalls pointers to all objects of a given port (0/1/2 - 3-33 on a GX) onto the stack, ignoring the R/W status of that port.

```
RLIB            ( :%port:%LID           --> lib                 )
                ( %LID                  --> libn %portn ...     )
```

Recall lib; the 1st case recalls a library from a given port, the 2nd case searches ports 0,1,2 (followed by ports 3-33 on a GX) for

libraries with %LID, returning all found libraries and the port numbers where they're stored.

Note: On a SX this command actually returns pointer to libraries (like RPORT), if you recall a lib and try to purge it while it's on the stack, you'll get a 'Object in use' error. Execute NEWOB or store it into a variable first.

PGLIB (:%port:%LID -->)
(%LID -->)

Purge lib; the 1st case works like :%port:%LID PURGE, in the 2nd case the ports are searched in order 0,1,2 (followed by ports 3-33 on a GX) for an active library with %LID. The difference to PURGE: if the library is attached to the home directory, it's detached before purging. On a SX: Also if there is an inactive library with the same LID in any other port, it becomes active and is attached to the HOME directory (if flag 5 is set, its config code is executed - see STLIB below).

STLIB (lib %port -->)

Store lib into port; there're a few differences to STO:
- The library is installed full; a warmstart isn't neccessary and thus not initiated at the next power cycle. All warmstart volatile variables (stack, PICT) remains intact.
- The library last stored is visible to the '48 (in case of having a library with the same LID installed in another port).
- If flag 5 is clear, the library is attached simply to the home directory.
- If flag 5 is set, the config code of the library is executed under warmstart conditions. Usefull for testing a config code.

Note: This command is currently disabled on a GX !

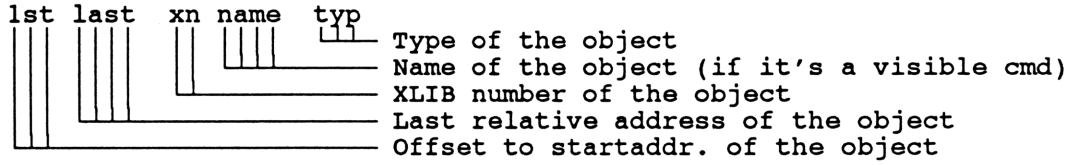
ACLIB (:%port:%LID -->)

Activate library. You can install libraries with the same LID in different ports, but only one will be visible to the '48 at the time. During warmstarts the ports are searched in order 2,1,0 (most cases), ie. the library stored in the port with the highest number will be active. ACLIB allows you to switch to any other library with the same LID at runtime, the effect is immidiate. ACLIB 1st detaches the LID from HOME, sets the new priority and than a) attaches the LID to HOME again if flag 5 is clear, or b) runs the library config code if flag 5 is set (see STLIB above).

Note: This command is currently disabled on a GX !

LIBp (%LID --> $)

Returns a detailed layout of a library. The map starts with the title (if exist), followed by the 1st and last address of the lib and the LID. The remainder lists the contents of the lib, one line of information for each XLIB entry. Structure of a line:



The list is sorted by offset. Try 1221 LIBp or 2 LIBp.

Note: If you find unexpected 'holes' between two XLIBs (> 10 nibbs) or XLIBs embedded in other XLIBS, the library wasn't generated using USRLIB.EXE or D->LIB; do not use D->LIB for recreating it because the 'holes' can hold essential data..

Note: This command is currently disabled on a GX !

INSTp (--> { %LIDn .. %LID1 })

Returns a list of all libraries attached to the current directory, { } if none.

Note: You can PURGE libraries even if they are attached to a subdirectory. INSTp can be used to find such zombie references.

LIBSp (--> { %LID1 .. %LIDn })

Returns a list of all libraries currently installed on your '48.

Note: We didn't use the '?' postfix because normally it marks a routine for returning a flag in RPL. The JARGON file, v2.9.9, 01 APR 1992 states:

3. The '-P' convention: Turning a word into a question by appending the syllable 'P'; from the LISP convention of appending the letter 'P' to denote a predicate (a boolean-valued function). The question should expect a yes/no answer, though it needn't.

At dinnertime:
Q: "Foodp?"
A: "Yeah, I'm pretty hungry." or "T!"

At any time:
Q: "State-of-the-world-P?"
A: (Straight) "I'm about to go home."
A: (Humorous) "Yes, the world has a state."

so we used 'p' for marking routines returning information ;-)

fEVAL (ob --> ?)

Works like EVAL, but switches the display off first. Speeds up evaluation by ~11%. In case of an error or the ob has finished execution, the display is switched on again. Not very useful, if ob prompts for input..

Things to Notice

The library stucture is 'flat', so don't try to include subdirectories in a library, it can end up in a memory lost.

Not all program objects that execute correctly from global variables are directly convertable into libraries. Here are some pitfalls:

- Since a library cannot be modified, no library command may be the target of a STO or PUT operation.

- XLIB names are not usable in all contexts in which global names are valid arguments. This can cause constructs that reference a named object to fail. For example,

```
'A' 5 GETI
```

where A is a list will not work when A is converted to an XLIB name.

Instead use

```
A 5 GETI
```

- XLIB names are not valid as formal variables in algebraics, or as the independent variable for plotting or solving.

- \->STR applied to a global name that is converted to a 'hidden' library command (see $HIDDEN) returns a null string.

If any visible command starts with the seqence '\<< \->' or '\->' it's marked in the library as a valid command for algebraics. If you press its associated softkey in ALG entry mode, you'll get 'name()'.

Multiple occurences of variable names results in an incorrect library because only the contents of the first one is picked up.

D->LIB needs ~(1.2 * size_of_source_directory) bytes to be free to generate a library.

The time D->LIB needs for doing a job depends mainly on the total number of commands included in the resulting library. Eg. Raymond runs D->LIB on a ~60kb directory containing ~300 variables; D->LIB needs ~15min on a rev A '48 to make the library (not in FAST-mode). Of course, this is quite faster than using USRLIB.EXE !!!

Reassembling a split library may be dangerous if the original library was not generated using USRLIB.EXE or D->LIB. There is no guarantee that the result will work properly - even if no code changes are done.

USRLIB.EXE generates a link table entry for the configuration program; if you split such a library with L->DIR, you'll get the configuration code twice, the first one stored in $CONFIG, the second one stored in a variable of the generated directory. Purge the variable before using D->LIB. You also can use LIBp to see the second reference to the config code.

Quick Reference Guide

Commands

Page 1221.01

D->LIB	(	--> lib	) build library
L->DIR	(%	--> dir	) split library
MCFG	(	-->	) make config program
ML->D	(	--> prg	) make libdat-> handler
MD->L	(	--> prg	) make ->libdat handler
OB->	(ob	--> ?	) split object

Page 1221.02

->DIR	(meta(ob,id)	--> dir	) make directory
->PRG	(meta(ob)	--> prg	) make program
->XLIB	(% % | # #	--> xlib	) make XLIB
->ARR	(meta(ob)	--> arry	) make array
->ALG	(meta(ob)	--> alg	) make algebraics
->LD	({} %	--> libdta	) make library data

Page 1221.03

```
    ->BAK       ( ob $          --> bak          ) make backup
    ->ID        ( $             --> id           ) make identifier
    ADRp        ( ob            --> ob #         ) get address of obj
    $romid      (               --> id           ) get id '$ROMID'
    $visible    (               --> id           ) get id '$VISIBLE'
    $title      (               --> id           ) get id '$TITLE'
```

Page 1221.04

```
    $config     (               --> id           ) get id '$CONFIG'
    $vars       (               --> id           ) get id '$VARS'
    $hidden     (               --> id           ) get id '$HIDDEN'
    $message    (               --> id           ) get id '$MESSAGE'
    LBCRC       ( lib | bak     --> lib' | bak'  ) recalculate CRC
    RNLIB       ( lib $         --> lib'         ) rename library
```

Page 1221.05

```
    CHLID       ( lib %         --> lib'         ) change LID
    RHASH       ( %             --> C#           ) get hash table
    RLINK       ( %             --> C#           ) get link table
    RCFG        ( %             --> prg          ) get config code
    RMSG        ( %             --> arry         ) get message table
    RTITLE      ( %             --> $            ) get title
```

Page 1221.06

```
    RPORT       ( %             --> ob ...       ) recall port
    RLIB        ( %             --> lib % ...    ) recall library(s)
    PGLIB       ( % | :%:%      -->              ) purge library
    STLIB       ( lib %         -->              ) store library
    ACLIB       ( :%:%          -->              ) activate library
    LIBp        ( %             --> $            ) get library layout
```

Page 1221.07

```
    INSTp       (               --> {}           ) get installed LIDs
    LIBSp       (               --> {}           ) get all LIDs on 48
    fEVAL       ( ob            --> ?            ) fast EVAL
```

Flag Usage

5 Set: STLIB executes config code of passed library after installation.
ACLIB executes config code of activated library.
<-LIB->s config code displays a (c) notice.
Clear: STLIB and ACLIB are simply attaching the handled library to the HOME directory.
<-LIB->s config code doesn't display (c) notice.

6 Set: L->DIR places the $control variables at the end of the generated directory.
Clear: L->DIR places the $control variables at the beginning of the generated directory.

7 Set: L->DIR generates a $HIDDEN but no $VISIBLE variable.
Clear: L->DIR generates a $VISIBLE but no $HIDDEN variable.

-13 Set: D->LIB/L->DIR switch the display off while processing a directory/library (fast mode). [Also, the ABOUT screen is displayed during a warmstart. -jkh-]
Clear: D->LIB/L->DIR are leaving the display on while working.

Error Messages

"Missing $ROMID"
    $ROMID is not defined in the current directory

"$ROMID Not Real/Binary"
    $ROMID doesn't contain a % or HXS object

"$ROMID Out of Range"
    a) the value of $ROMID is > 2047
    b) a % > 2047 was passed to CHLID

"$CONFIG Not a Program"
    Because the stack must not change during warmstarts, $CONFIG must contain a program

"$HIDDEN Not a List"
"$VISIBLE Not a List"
"$VARS Not a List"
"$MESSAGE Not a List"
    $XXX must contain a list

"Found ID Name>16 Chars"
    D->LIB found a visible command name which is > 16 chars in size

"Found 0-ID"
    D->LIB have found a visible command name with the size 0

"Won't work on a G"
    The initiated operation can't be executed on a '48G(X).

Ordering Information

The sources of the <-RPL-> and the <-LIB-> libraries consists of more than 18000 lines of RPL/assembler code (that's more than 300kb of text !) and we have spent our free time for more than two years in writing these, so we decided to release this version as GiftWare - ie. you can use these as long as you like but only for developing non-comercial software and only as a private person.

If you think <-LIB-> is useful and that the authors deserve to be rewarded for the time they have invested in developing this toolkit, feel free to send one of us (or both ;-) any sort of gift (even only a postcard will be welcome).

If your gift covers the expense sending you a disk via SnailMail (costs are: the disk, an envelope, the stamp and our time -> ~$25 or something equivalent), we will send you the latest *whole* version of <-LIB-> on a MesS-DOS formatted disk (including the <-RPL-> library branded with your name in the startup message). Don't forget to include your name, address and the disk size you prefer.

Developing commercial software using <-LIB-> requires registration via the GiftWare concept; companies must send us at least a $50-worth gift for registration.

Contact addresses:

Detlef Mueller
Bellerbek 33
D-22559 Hamburg
Germany

Raymond Hellstern
Liebigstr. 8
D-30163 Hannover
Germany

e-mail: detlef@dmhh.hanse.de

JAZZ V4.0

System RPL and Machine Language Development Library

(c) 1995 by Mika Heiskanen & Jan Brittenson

CONTENTS
========

1. Introduction
    1.1 Copyrights & Ackowledgements
    1.2 The Jazz Library
    1.3 Installing & Deleting the Library

2. Jazz Commands
    2.1 The System RPL/Machine Language Assembler
    2.2 The System RPL/Machine Language Disassembler
    2.3 The System RPL Debugger
    2.4 The Machine Language Debugger
    2.5 The System RPL Stack
    2.6 The Entries Catalog
    2.7 The System RPL/Machine Language Editor
        2.7.1 The Viewer
        2.7.2 The Small Font
        2.7.3 The Medium Font
    2.8 Entries Table Utilities

1. Introduction
===============

1.1 Copyrights & Acknowledgements

Extra credits & acknowledgements:

Jan Brittenson	DB program is originally from Jan's MLDL library and is still copyrighted by him. The mnemonics have been changed to the ones used by HP + some minor changes has been made.
Mario Mikocevic	The small 4x6 font + the basis for the machine language instruction disassembler.
Will Laughlin	Backward search in ED
Rick Grevelle	The medium 6x8 font.
Detlef Mueller & Raymond Hellstern	Inspiration through RPL48 package.
Dan Kirkland	Sorting the default tables sensibly
Jens Kerle	Bug fixing
Cary McCallister	For answering.

Beta testing & suggestions:

Seth Arnold	Bill Levenson
Douglas Cannon	Tom van Migem
Carlos Ferraro	Mario Mikocevic
Rick Grevelle	Detlef Mueller
Joe Horn	Richard Steventon
Boris Ivanovich	Kurt Vercauteren
Jens Kerle	Vladimir Vukicevic
Dan Kirkland	Christ van Willegen
Jeoff Krontz	Stefan Wolfrum
Will Laughlin	

+anyone else who I may have forgotten

1.2 The Jazz Library

The Jazz library provides commands for assembling, disassembling and debugging both system rpl and machine language. This document describes only the provided commands, not the languages themselves. For information on the languages please refer to the tools package published by HP, especially the files RPLMAN.DOC, RPLCOMP.DOC and SASM.DOC. Familiarity in the fundamentals is assumed from now on.

Following files available from hpcvbbs.external.hp.com are recommended:

dist/ms-dos/tools.exe	HP Tools
dist/hp48g/programming/entries/ent_srt.zip	Sorted Entries
dist/hp48s/programming/entries/entries.zip	Address sorted entries
dist/unix/sadhp105.zip	Unix Disassembler

GNU Tools are available from srcml.zems.fer.hr (IP 161.53.64.254)

pub/hp48/tools2.0.4.zip

1.3 Installing & Deleting the Library

To install the Jazz library:

SX: Plug the HP48 Hacker's ROM into port 1 or port 2.
GX: Plug the HP48 Hacker's ROM into port 1.

To remove the Jazz library:

Unplug the HP48 Hacker's ROM.

2. Jazz Commands

2.1 The System RPL/Machine Language Assembler

As opposed to the tools provided by HP Jazz provides only one command to assemble source code. The assembler assumes the source code to be RPL unless a switch is made via special tokens.

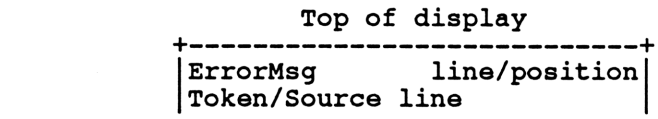
Command: ASS
Stack: ($ --> ob)
Description: Assemble source string
Keys: ON key aborts assembly
User flags:
1 - Report mode on (slows down assembly considerebly!)

Errors:
Special error trap simulation is used to enable using

the small font for error messages and thus get more information of the reason for the error.
For the small font to be used the current program (possibly the kernel) must use SysErrorTrap to trap errors. If the trap found is different error handling is left to the found error trap and extra information of the reasons for the error will be lost, only the actual error number is provided for the trap.
For programmers conveniece also error traps that start with ":: NOP" cause showing the full error message.

When possible ASS will also output the error position to the stack so that you can immediately edit the error line/token.

Display when the error is trapped:



Comments in RPL mode are:

"*" at the start of the line marks the entire line to be a comment.
"(anything)" surrounded by whitespace is considered a comment.

RPL Assembly Mode Tokens

```
;                --> SEMI
{                --> DOLIST
}                --> SEMI
::               --> DOCOL
SYMBOL           --> DOSYMB
UNIT             --> DOEXT

# hhhhh                  --> system binary
#hhhhh                   --> system binary (from ROM if possible)
ddddd                    --> system binary
PTR hhhhh                --> pointer
ACPTR hhhhh hhhhh        --> access pointer, G/GX only
ROMPTR hhh hhh           --> rom pointer object
ddd.dd                   --> real number
% ddd                    --> real number                 (*)
%% ddd                   --> long real number            (*)
C% ddd ddd               --> complex number              (*)
C%% ddd ddd              --> long complex number         (*)

(*) ddd can also be -Inf, Inf or NaN

HXS      <len> <hh.h>    --> hex number
GROB     <len> <hh.h>    --> grob
LIBDAT   <len> <hh.h>    --> library data
BAK      <len> <hh.h>    --> backup object
LIB      <len> <hh.h>    --> library object
EXT1     <len> <hh.h>    --> external type 1, S/SX only
EXT2     <len> <hh.h>    --> external type 2
EXT3     <len> <hh.h>    --> external type 3
EXT4     <len> <hh.h>    --> external type 4
ARRY     <len> <hh.h>    --> array
LNKARRY  <len> <hh.h>    --> linked array
CODE     <len> <hh.h>    --> code object
NIBB     <len> <hh.h>    --> misc nibbles
```

If <len> is zero then <hh.h> the resulting object will consist of the prolog indicated by the first token and "00005" length field. (NIBB indicates no prolog so len must be non-zero)

```
$ "<string>"            --> string
"<string>"              --> string
ID <string>             --> identifier object
LAM <string>            --> lambda identifier object
TAG <string> <..>       --> tagged object
CHR <char>              --> character object

xROMWORD                --> pointer or ROMPTR depending on the
                            library number the command belongs to
                            ( For example "xDUP" compiles to a
                            pointer but "xASS" to a rom pointer)
INCLOB <name>           --> Include object stored to a variable
INCLUDE <name>          --> Include source code
DEFINE <token> <string> --> Define substitute for token
                            Substitution will not be done inside
                            multipart tokens, for example
                            "ID <token>" is not allowed
CODE <newline>          --> Starts machine language assembly
ASSEMBLE                --> Starts machine language assembly
```

ML Assembly Mode Mnemonics

Opcodes recognized in machine language assembly are all the normal opcodes indicated by SASM.DOC plus the next new ones as implemented in GNU Tools:

```
        LCSTR   \ASCII\         Reversed LCASC
        LASTR   \ASCII\         Reversed LAASC
        CSTRING \ASCII\         NIBASC with 0-byte terminator
        ABASE   expr            Sets allocation counter to address
                                specified by <expr>.
label   ALLOC   expr            Allocates <expr> nibbles for label
                                at the allocation counter, then
                                increases allocation counter by <expr>
```

MAKEROM

ASS can be used to assemble MAKEROM source code with the following tokens:

```
xROMID #hhh             Defines hex library number.
xROMID dec              Defines decimal library number
xTITLE <title>          Defines title to be rest of the line.
                        If title is missing then nulltitle is used.
xCONFIG <label>         Defines the location of the configuration
                        object via a label. If label is missing then
                        no config is taken to exist.
xMESSAGE <label>        Defines the location of the message table
                        via a label. If label is missing then no
                        message table is taken to exist.
EXTERNAL <label>        Defines label to be external. Order of
                        introduction determines the command number
                        of the commands in the library so that
                        visible commands will be first, then
                        nullnames.
xNAME <label>           Specifies location of a visible command.
                        Name: "xlabel"  Hash: "label"
sNAME <label> <hash>    Specifies location of a visible command.
                        Name: "label"   Hash: "hash"
hNAME <label>           Specifies location of a 'visible' command.
                        Name: "label"   Hash: null
NULLNAME <label>        Specifies location of a hidden command.
                        Name: "label"   Hash: none
tNAME <label> <hash>    Specifies secondary hash for command.
```

xROMID and xTITLE must be used at the start of the source code, to be specific before any actual code has been output. Also xROMID must come before any other MAKEROM token.

xCONFIG and xMESSAGE declarations can be anywhere (or absent), but a suitable location is after the xROMID and xTITLE declarations.

EXTERNAL declaration is needed if the corresponding command is used before its location is specified by its NAME declaration. Suitable location is after the header declarations, and it is probably best to declare all commands.

xNAME, sNAME, hNAME and NULLNAME specify command location, thus they should be right in front of the object they define as a command. All but NULLNAME also define the romid/cmd header field properly and thus require a propfield, typically the value 8 to mark a regular command and 000 to mark a regular function, for other values please refer to entries.srt or other documents. Above also define a symbol '~label' having a 6 nibble value, low 3 nibbles containing the romid and high 3 nibbles the command number.

The internal menu display routines stop showing library menus if a command with no hash is found. Thus any command declared after a hNAME will not be shown in the library menu, but will of course have typable/disassemblable command names as usual if so specified by NAME tokens.

tNAME can be used anywhere after the declaration of the romp the secondary hash is assigned to, a suitable location is right after the corresponding NAME location declaration. Note that tNAME can be used to declare names for NULLNAMEs, thus providing easy access to low level subroutines if needed. One command can have several secondary names.

Note that INCLOB does not do any ID --> ROMP conversion work on the included object like the common DIR --> LIB library builders do.

Example: Jazz MAKEROM source would start like this:

```
xROMID  992
xTITLE  Jazz v4.0 Fin'95 10.06.95 mheiskan@gamma.hut.fi
xCONFIG JazzCfg
xMESSAGE JazzMsg

EXTERNAL xFNT1          ( Fonts always come first in Jazz )
EXTERNAL xFNT2
EXTERNAL xASS           ( User ASS command )
EXTERNAL Assemble       ( Main assembler code object )
[..]
EXTERNAL UnShowSel!     ( Low level subroutines of SSTK )
EXTERNAL >SelPict!

LABEL JazzCfg           ( Configuration object )
:: 992 TOSRRP ;

LABEL JazzMsg           ( Message table )
ARRY hhhhh hhh..h

NIBB 1 8                ( Easier than ASSEMBLE CON(1) 8 RPL )
xNAME ASS               ( Note: the EXTERNAL declarations )
:: CK1 ... ;            ( declared fonts to come before ASS )
[..]                    ( when assigning command numbers )
```

Missing Tokens and Mnemonics

The following ones for various reasons do not behave as documented in SASM.DOC or GNU Tools:

Not implemented:

```
IF, ELSE, ENDIF + any other conditional assembly mnemonic
MACRO           \
ENDM                    Macros not implemented
EXITM           /
CLRFLAG expr    \       Flags not implemented
SETFLAG expt    /

ABS     expr            Not Implemented
RDSYMB  file            ..
CHARMAP file            ..
Dn=HEX  hh.h            ..
GOSHORT label           ..
JUMP    label           ..
INC(n)  label           ..
LINK    label           ..
SLINK   label           ..


NIBBIN  bb..b           ..      (GNU Tools Opcodes)
NIBGRB  bb..b           ..
HEX(n)  hh.h            ..
HEXM(n) hh.h            ..
ASC(n)  \ASCII\         ..
ASCM(n) \ASCII\         ..
```

Modified Tokens and Mnemonics

Behaviour changed:

```
TITLE   text            Text shown on line 1, line 2 cleared
STITLE  text            Text shown on line 2
MESSAGE text            Text shown on line 1, line 2 cleared

D0=D0+  expr            \
D0=D0-  expr             Allow values between 1 - 256, and
D1=D1+  expr             generate multiple opcodes if needed.
D1=D1-  expr            /
```

Ignored:

EJECT, REL, LIST, LISTM, LISTALL, UNLIST

Expressions in machine language

Factors:

```
        #hh.h           hex integer
        %bb.b           binary integer
        dd.d            decimal integer
        =symbol         global symbol (RPL.TAB checked too)
        :symbol         local symbol
        symbol          local symbol
        *               PC counter
        + - ++ --       Local labels
```

Operators:	Priority:	
^	9	
*	8	
/	8	
%	8	(modulo)
+	7	
-	7	
&	5	(and)
!	4	(or)

Note in particular that ascii factors are not implemented. As in sasm all symbols must be surrounded with parentheses when operators are used.

Label generation in machine language

Symbols +, ++, - and -- are location dependant symbols and refer to the next/previous defined value of the corresponding symbol. Example:

```
        GOSUB   +       ----+
        CSTRING 'Foobar'    |
+       C=RSTK          <---+
        D0=C                |
-       A=DAT0  B       <=======+
        DAT1=A  B               |
        ?A=0    B               |
        GOYES   +       ----+   |     End of string, done
        D0=D0+  2           |   |
        D1=D1+  2           |   |
        GONC    -       ====|===+
+       ...             <---+
```

Note: Symbol ++ does not refer to the 2nd + coming up, but the next ++ label coming up, and similarly for --.

INCLOB Special Features

INCLOB will behave differently from code than from rpl. Depending on the type of the object being included it is either included entirely or the leading nibbles (prolog + possible data fields) are skipped for following object types:

Prolog	Skip	Prolog	Skip	Prolog	Skip
DOCODE	10	DOEXT0	10	DOGROB	20
DOCSTR	10	DOEXT2	10	DOARRY	5
DOHSTR	10	DOEXT3	10		
		DOEXT4	10		

Warnings

A=A+CON fs,expr and similar instructions allow single nibble fields (WP,P,S,XS) while SASM errors. These instructions behave badly due to a hardware bug but are supported by ASS for those who know how to safely use the commands.

Symbol/label handling is quite secure but allows more than SASM. For example external values have no significance, thus for example (=GETPTR)-(=SAVPTR) is valid in Jazz. Relative values are followed and are significant for most opcodes. For example 'D0=D0+ label' will error. Note especially that (label1)-(label2) is absolute.

= EQU and ALLOC require their expression fields to be resolvable on the first pass.

2.2 The System RPL/Machine Language Disassembler

Several disassembler commands are provided for different purposes.

```
Command:        DIS
Stack:          ( ob --> $ )
Description:    Disassemble object. If stk1 is a pointer to a ROM
                address only the pointer will be disassembled.

Command:        DISXY
Stack:          ( hxs_address hxs_end_address --> $ )
Description:    Disassemble memory area.
                Guesses start mode, switches mode during
                operation if necessary.

Command:        DOB
Stack:          ( ob | #address | hxs_address | "entry" --> $ )
Description:    Disassemble memory area.
                Guess start mode and end address.

Command:        DISN
Stack:          ( hxs_address %N --> $ )
                Disassemble memory area as machine language only.
```

Common features for the disassembler commands:

ON key aborts disassembly.

User flags:

2 - disable guess mode
4 - disable machine language disassembly for DIS
5 - disable tabulator, use spaces instead
6 - force generated labels on their own rows

In guess mode the disassembler will try to guess data structures embedded in machine language. Currently only the following types are recognized:

```
        GOSUB   +
        REL(5)  +               Optional leading size indicator
        BSS     expr            Data is all zeros
+       C=RSTK

        GOSUB   +
        REL(5)  +               Optional leading size indicator
        NIBASC  \ASCII\         \ Possibly alternating and spanning
        CSTRING \ASCII\         / multiple lines.
+       C=RSTK

        GOSUB   +
        REL(5)  +               Optional leading size indicator
        NIBHEX  hh.h            Miscellanoues data
+       C=RSTK
```

The sufficient condition for an ascii guess to be successful is that the data area should consist mostly (75%) of common ascii characters. The ascii lines are splitted to CSTRINGs or by newline characters or so that the maximum length will be 40 chars.

Warnings:

- DOB, DISXY and DISN disassemble areas of memory instead of well defined objects. Using these commands to disassemble memory in the temporary object area is dangerous since a possible garbage collection during run-time can move the memory being disassembled.

Use only DIS to disassemble objects in tempob!!!
- Composite history is tracked up to 64 levels. If that is exceeded a ";" may be output when "}" is due.
- As opposed to the assembler the disassembler cannot handle hidden hash tables, thus some named ROMPTRs will be disassembled to "ROMPTR xxx yyy".
- Label values are guessed for Dn=(2) and Dn=(4) instructions if
  - Dn=(2) is likely to refer to the IO page
  - Dn=(4) is likely to refer to a RAM variable

  This works well for disassembling ROM but doesn't do well when disassembling a program that uses even-page method in its data allocation. Benefits are clearly greater though.

2.3 The System RPL Debugger

```
Command:        SDB
Stack:          ( seco | id | lam | romp --> ? )
Description:    Start debugging program indicated by stack level 1.
```

If the srpl debugger is already running then SDB command will only show the SDB menu:

->SST	- Single step next command If right-shifted then single-steps rest of the stream as a single unit
->IN	- If possible then enter the program referred to by the next command, else single step command.
SNXT	- Show next commands on status area Pressed for the second time shows return stack
SST->	- Start continuous ->SST mode, subsequent presses toggle slow/fast mode, eg whether stack display is updated after each command or not. Any other key aborts continuous evaluation
IN->	- Start continuous ->IN mode.
DB	- Start DB on next code object
xKILL	- The HP48 KILL command (xCONT can be evaluated through LS+ON keys)
SKIP	- Skip next command. If right-shifted then skips rest of current stream, eg executes a SEMI command.
SEXEC	- Execute stk1 as the 'next' command.
SBRK	- Set breakpoint object to STK1. If right-shifted clears breakpoint object.
LOOPS	- Browse loop environments. Up/Down to scroll, any other key to exit. If right-shifted dumps topmost environment to the stack.
LAMS	- Browse lam environments. Up/Down to scroll lams, left/right to decrease/increase environment. Any other key to exit. If right-shifted dumps topmost environment to the stack.
IN?	- Toggle ->IN mode to never enter into secondaries, only into IDs/LAMs/ROMPTRs when allowed. Prevents the debugger from entering into ROM subroutines

during continuous debugging.

Note that SDB is meant for debugging system rpl, not user rpl. Thus some user rpl commands will not be single stepped right when using SDB. One example is xHALT, for which the substitute xSHALT is provided. Note that SDB must be running before SHALT works.

Warnings:

Debugging system-rpl is very hairy and undoubtedly SDB cannot debug some lesser known commands correctly. If such commands are found SDB can even cause a crash and memory loss. This is unfortunately unavoidable since there really is too much code in ROM to worry about. SDB should manage to debug all normal programs though.

SDB either enters commands or executes commands (by emulation if necessary). None of the interactive commands in HP48 ROM are emulated, most importantly PolOuterLoop. To emulate POLOuterLoop you need to insert SHALT commands into the display objects or whatever you want to debug, then start SDB, then use CONT to reach the point of the SHALT command.

LOOPS and LAMS displays are pretty lame, I'll try to improve them later.

2.4 The Machine Language Debugger

Command: DB
Stack:
(id -->)
(romp -->)
($entry -->)
(#address -->)
(hxs_address -->)
(code -->)
Description: Debug machine language

Screen Keys:

[A]	- Screen 1 (general registers)
[B]	- Screen 2 (registers A-D)
[C]	- Screen 3 (registers R0-R4)
[D]	- Screen 4 (RSTK)
[E]	- Screen 5 (memory dump)
[F]	- Screen 6 (machine language disassembly)
[MTH]	- Screen 7 (breakpoints)
[VAR]	- Screen 8 (watchpoints)
[ ]	- Update display
[EEX]	- View PICT (if it exists) as long as EEX is down

Arguments:

[0] - Start inputting argument.

[0-9A-F]	add digit
[DEL]	abort input
[BS]	delete last digit
[+/-]	negate arg

Movement keys:

[NXT]	- Skip instruction	(or ARG instructions)
[left]	- PC=PC-1	(or -ARG)
[right]	- PC=PC+1	(or +ARG)
[up]	- PC=PC-16	(or -16*ARG)
[down]	- PC=PC+16	(or +16*ARG)
[.]	- Set mark to PC	(or to ARG)
[+/-]	- Swap PC and mark	
[ENTER]	- If ARG then set PC = ARG	

Debug keys:

```
        [+]     - Single step (ARG) instructions
        [-]     - Single step (ARG) instructions,
                  debug GOSUBs as a single instruction
        [*]     - Single step (ARG) instructions with display update.
                  If no ARG then sets ARG to #FFFFF.
        [/]     - Single step (ARG) instructions with display update,
                  debug GOSUBs as a single instruction.
                  If no ARG then sets ARG to #FFFFF.
        [EVAL]  - Continue until end or breakpoint

        [SIN]   - Save current registers
        [COS]   - Save registers with the saved ones
                  (first save is done at startup)

        ARG + [SIN]     = clear cycle counter 1
        ARG + [COS]     = clear cycle counter 2
```

Exit keys:

```
        [DEL]   - Restore registers & exit
        [BS]    - Exit now
        [1/x]   - Exit via reset, press second time to confirm
```

Breakpoints:

```
        [PRG]   - Set breakpoint to ARG
        [STO]   - Set breakpoint counter to ARG
```

Options:

```
        [CST]   - Toggle option number (next key)

                3       - Ascii/hex mode
                4       - Shift memory dump by 1
                5       - Automatic switch between PICT/ABUFF
                          during debug
                6       - Disable/enable RPL.TAB and DIS.TAB
                7       - Opcode/cycles display
```

Sample screens which can be reasonably reproduced by:

```
        "#>HXS" DB      (or #59CCh DB)
```

Screen 1 - General CPU State (key [A])

```
Mnemonic.............................  GOSUBL  SAVPTR
Opcode...............................   8E4CD0
PC, P, Carry, Hex/Dec mode, ST......  @:059D1 P:0 CH ST:298
A.A and C.A..........................  A:059CC C:BF4F8
B.A, D.A, and HST....................  B:8883E D:0AF58 HST:2
D0 and 6 bytes @D0...................  D0:E8EF8/CCD205700074
D1 and 6 bytes @D1...................  D1:BF4FD/000000000000
Top 3 levels of RSTK.................  RST:00000:00000:00000
```

Screen 2 - Arithmetic registers (key [B])

```
Mnemonic.............................  GOSUBL  SAVPTR
Opcode...............................   8E4CD0
PC, P, Carry, Hex/Dec mode, ST......  @:059D1 P:0 CH ST:298
Register A...........................  A:6C4475C79A7059CC
Register B...........................  B:000000000008883E
Register C...........................  C:36000000077BF4F8
Register D...........................  D:000000000000AF58
Top 3 levels of RSTK.................  RST:00000:00000:00000
```

Screen 3 - Data registers (key [C])

```
Mnemonic.............................  GOSUBL  SAVPTR
Opcode...............................   8E4CD0
PC, P, Carry, Hex/Dec mode, ST......  @:059D1 P:0 CH ST:298
Register R0.........................  R0:60000000000409C1
Register R1.........................  R1:6C4475C79A7059D1
Register R2.........................  R2:6C4475C79A7DE599
Register R3.........................  R3:6C4475C79A700000
Register R4.........................  R4:10000000004BFA18
Top 3 levels of RSTK................  RST:00000:00000:00000
```

Screen 4 - Return stack (key [D])

```
Mnemonic.............................  GOSUBL  SAVPTR
Opcode...............................   8E4CD0
PC, P, Carry, Hex/Dec mode, ST......  @:059D1 P:0 CH ST:298

RSTK levels 0 and 4.................  RST0:00000 RST4:00000
RSTK levels 1 and 5.................  RST1:00000 RST5:00000
RSTK levels 2 and 6.................  RST2:00000 RST6:00000
RSTK levels 3 and 7.................  RST3:00000 RST7:00000
```

Screen 5 - Memory dump (key [E])

```
Locations 59A0-59AF.................  059A0:56113680913420CC
Locations 59B0-59BF.................  059B0:4E0156716FCC56FD
Locations 59C0-59CF.................  059C0:015B38D5E0101D95
Locations 59D0-59DF.................  059D0:08E4CD08E46C0101
Locations 59E0-59EF.................  059E0:D230574911191443
Locations 59F0-59FF.................  059F0:4E4A201101311456
Locations 5A00-5A0F.................  05A00:12280A50143174E7
Locations 5A10-5A1F.................  05A10:8E58D01311741431
```

current location is indicated by an inverse digit.

Screen 6 - ML Instruction Stream (key [F])

```
PC, P, Carry, Hex/Dec mode, ST......  @:059D1 P:0 CH ST:218
Next 7 instructions.................   D1: GOSUBL  SAVPTR
                                       D7: GOSUBL  POP#
                                       DD: R1=A
                                       E0: C=0     A
                                       E2: LC(1)   5
                                       E5: GOSUB   MAKE$N
                                       E9: C=R1
```

The next instruction is the one displayed in reverse. Currently, it will always appear at the top.

Screen 7 - Breakpoint Table Screen (key MTH)

```
Breakpoint #1.......................   1:6100 +02
Breakpoint #2.......................   2:6104 -02
Breakpoint #3.......................   3:613A  00
Breakpoints #4-#8: not used.........   4:0000  00
                                       5:0000  00
                                       6:0000  00
                                       7:0000  00
                                       8:0000  00
```

Any breakpoints at the current location are displayed in reverse.

Warning:
DB uses D0 for its own purposes, thus you cannot debug code that modifies the rpl return stack nor the current stream.

2.5 The System RPL Stack

The SSTK command starts a new kernel which is a modified version of the internal one. To exit SSTK just execute SSTK again.

Modifications to the internal kernel:

- Stack has 5 lines, including the interactive stack.
- Flag 3 toggles the stack decompiler:
  Set: Use internal decompiler
  Clear: Use a system rpl decompiler
- Multi-line mode is not supported.

2.6 The Entries Catalog

EC command is a browser for the entry tables. Since the entries are listed in address sorted form both RPL.TAB and DIS.TAB are needed to run EC. [They are in **Library 993** in the HP48 Hacker's ROM. -jkh-]

Keys are:

Up Arrow	- Up one entry
Down Arrow	- Down one entry
LS + Up Arrow	- Up one page
LS + Down Arrow	- Down one entry
RS + Up Arrow	- Jump to first entry
RS + Down Arrow	- Jump to last entry
Right Arrow	- View the contents for selected entry with VV
Alpha	- Input find string (entry name grep)
F	- Input find string (entry name grep)
NXT	- Find next match
LS + NXT	- Find previous match
EEX	- Toggle grep mode (show only matches)
ENTER	- Push entry to stack as :name:address
LS + ENTER	- Push entry address to stack
RS + ENTER	- Push entry name to stack
0-9	- Find entry starting with input address Use 0-9A-F to input a more specific address.
ON	- Exit browser
+/-	- Toggle beep on/off

2.7 The System RPL/Machine Language Editor

ED is an editor intended for editing rpl and machine language source code. ED makes no duplicate of the edited string if it is in temporary object area, thus enabling editing very large strings. Note that this implies that no backup of the original string is kept!

Note that ED is very fast but since it supports the tabulator it has to do special calculations whenever the display is scrolled. Thus scrolling the display when very long lines are present can be quite slow.

As a hopefully useful feature ED will accept an optional cursor position argument on stack level one. Thus if ASS gives you the error position you will be able to jump to that position immediately.

Most of the normal character keys are in their normal places, others can be fetched via the special character browser.

ED also allows an alternate 4x6 font to exist in variable 'FONT.ED'. No checks are made on the correct format, but obviously the just starting ED is an easy way to see if there is a problem.

Special keys having different definitions are mostly in the non-alpha plane. The NS,LS,RS planes are defined as follows:

BSTART	BEND	BCOPY	BDEL		FIND REPL? REPLALL
ARG? ROW? POS?	 CHR? CHRCAT	MEXEC MSTART MEND	REVERSE TOHEX TOASC	UP PGUP TOP	NEXT PREV STATUS
''	STK RCLSTK1		LEFT LSTART	DOWN PGDN BOTTOM	RIGHT LEND
GOTO GOLONG GOVLNG	GOSUB GOSUBL GOSBVL	GOYES GONC GOC		^ CNTRINI CNTR	DFIND
EXIT	 ASS DOB	TOGBEEP TOGCASE TOGOVER	DEL DELLINE DELRGHT	BS	
alpha	7 SETMK7 GOMK7	8 SETMK8 GOMK8	9 SETMK9 GOMK9	/ () #	
lshift	4 SETMK4 GOMK4	5 SETMK5 GOMK5	6 SETMK6 GOMK6	* [] ASS_RPL	
rshift	1 SETMK1 GOMK1	2 SETMK2 GOMK2	3 SETMK3 GOMK3	- <<>> ""	
REDISP OFF	0 GOMK0	. , NEWLINE	SPC TAB TAB	+ {} ::	

Explanations:

TOGBEEP - Toggle beep on/off. Default value is taken from the system flag.
TOGCASE - Toggle lower/upper case characters.
TOGOVER - Toggle insert/overwrite mode.

DEL - Delete character under cursor
DELLINE - Delete line under cursor
DELRGHT - Delete characters to right of cursor.
BS - Backspace

BSTART - Set block start address
BEND - Set block end address
BCOPY - Copy block/cut to cursor position
BDEL - Delete block (copied to cut)

```
RCLSTK1 - Pop string from stk1 into cursor position

FIND    - Incremental search. Search is case sensitive if find string
          contains lower case characters.
REPL?   - Find/replace with verification
REPLALL - Replace all
NEXT    - Find next match
PREV    - Find previous match
DFIND   - Find matching delimiter for delimiter under cursor

MSTART  - Start defining macro key sequence
MEND    - End macro key sequence
MEXEC   - Execute macro key

ARG?    - Input repeat count for next key press
ROW?    - Input row to jump to
POS?    - Input position to jump to
CHR?    - Input character number to insert
CHRCAT  - Character browser, ENTER key echos chosen character to
          cursor position, ON key exits.

SETMKn  - Set mark <n>
GOMKn   - Jump to mark <n>
GOMK0   - Go to previous cursor position

TOHEX   - Convert block to hex nibbles (Suitable for NIBASC -> NIBHEX)
TOASC   - Convert block to asc nibbles (Suitable for NIBHEX -> NIBASC)
REVERSE - Reverse chars in block/word
ASS     - Assemble source code, if error occurs shows the error
          message and after a keypress jumps to the error position.
DOB     - Disassembles entry under cursor using DOB, spans a new
          editor to view the disassembly. After exit back to the
          original editor the disassembly will be in the clip (if
          memory allows) ready to be inserted into the text if so
          desired. Special cases:
            #hhhhh          --> view (like plain entry)
            Lhhhhh          --> view
            ROMPTR hhh hhh  --> view
            PTR hhhhh       --> view
            ID name         --> visit contents    (RCL+DIS+ED+ASS+STO)
            INCLOB name     --> visit contents    (RCL+DIS+ED+ASS+STO)
            INCLUDE name    --> visit text contents (RCL  +ED    +STO)
            GROB hhhhh hh.h --> view grob
STK     - Starts a normal SOL. Recursive EDs are allowed. Exit back to
          ED with CONT key. All internal markers except cursor
          position will be lost.
CNTRINI - Initialize counter variable. Number of digits used
          determines width of counter, possible leading  "#"
          determines a hex counter.
CNTR    - Insert counter into text and increment it. At start the
          width is initialized to 1 hex nibble, so for example
          pressing
                        [ARG?] 16 ENTER [CNTR]
          will produce "0123456789F"

Special keys during inputline:

        ENTER           - Input ok
        ON              - Cancel
        DEL             - Delete char
        BACKSPACE       - Delete previous char
        LT/RT           - Move left/right. During find input pressing
                          RT at the end of input will take the next
                          input char from the current match location,
                          thus making it easier to complete the match.
        NXT             - Next match during find input
        PREV            - Previous match during find input.
```

Special keys in alpha plane:

```
A LS -   =       ::\n;           (with indent checks)
A RS -   =       $ ""
A LS +   =       {\n}            (with indent checks)
A RS +   =       CODE\nENDCODE
A ENTER =        \n + indent the same way as the previous line
```

Notes:

Repetition and macro key execution can be aborted with the ON key. Repetition, macro save and macro execution are aborted automatically if an error occurs. Max length of a macro key sequence is 50 keys.

2.7.1 The Viewer

```
Command:        VV
Stack:          ( $|grob --> $|grob )
Description:    Simple string/grov viewer.
```

Keys when viewing a string:

```
Up/Down/Left/Right      = scroll display
PRG/STO/'/EVAL          = scroll onedisplay page
F/NXT                   = top/bottom
-                       = slow scrolling
+                       = fast scrolling (default)
ON/ENTER                = exit
```

Keys when viewing a grob:

```
Up/Down/Left/Right      = move grob
.                       = center grob
ON/ENTER                = exit
A-F                     = choose scroll speed 1-6
```

The grob viewer uses a grob! replacement with automatic cutting. Masking grobs less than 4 bits wide is not not properly implemented yet. Viewing is done on the text grob, thus the following will create a weird effect: :: ABUFF xVV ;

2.7.2 The Small Font

The small font is fixed to ROMPTR 3E0 0 under the name FNT1. The format of the font is in assembly:

```
CON(5)  =DOEXT0          * Library Data
CON(5)  256*6+5          * 256 characters, 6 nibbles each
NIBHEX  ......           * Char 00
NIBHEX  ......           * Char 01
...
NIBHEX  ......           * Char FF
```

2.7.3 The Medium Font

The medium font is fixed to ROMPTR 3E0 1 under the name FNT2. The format of the font is in assembly:

```
CON(5)  =DOEXT0          * Library data
CON(5)  256*16+5         * 256 characters, 6 nibbles each
NIBHEX  ................ * Char 00
NIBHEX  ................ * Char 01
...
NIBHEX  ................ * Char FF
```

2.8 Entries Table Utilities

```
Command:        EA
Stack:          ( $entry   --> hxs_addr )
                ( hxs_addr --> $entry   )
                ( ob       --> hxs_addr )
Description:
                Converts between entry name and its address.
                For other argument types the address of the object
                is given

Command:        RTAB
Stack:          ( --> $ )
Description:    Recalls RPL.TAB

Command:        DTAB
Stack:          ( --> $ )
Description:    Recalls DIS.TAB

Command:        RTB->
Stack:          ( --> $ )
Description:    Converts RPL.TAB into readable form

Command:        ->RTB
Stack:          ( $ --> $' )
Description:    Converts an entry list into RPL.TAB form
                Lines accepted are:
                [=]name[whitespace][EQU #]address\n
                 ~~~                ~~~~~~~

                optional           optional

                Note that the input should be sorted, no checks for
                that are done. Also the last character in the input
                string should be a newline character.

Command:        ->DTB
Stack:          ( --> )
Description:    Creates a DIS.TAB based on RPL.TAB, stores it
                to home directory.
```

LIBEX (G/GX ONLY!)

Library Explorer & Extractor, by Marc Vogel and Régis Dechesne.
Documentation by Joe Horn.

This is a terrific library-extraction tool. It is the only tool yet written that extracts a command AND ALL OF ITS EXTERNAL CALLS into a directory, thus automating the otherwise tedious task of making subsets of large libraries. It is also the only tool that optionally extracts all of a function's header, such as what it does when you press RULES, ISOL, and derivative; whether it's allowed in algebraics, what it does in the EquationWriter, and more.

It can break an entire Library or just one function. And it can break ROM Library and recover ALL external calls to other libraries (XLIBs).

Parameters for LIBEX

Option 1: Number of the Library to break
In this case the entire library is extracted.

Option 2: List of XLIB/Functions/Commands
Only the specified ones are extracted.
External calls are extracted if specified by SETPREF.

Note : if you want to break a ROM Library, set the Recover External XLIB option to ON.

RCLXLIB is just a XRCL that works with ONE XLIB function put in a list.

Note: The first SETPREF option (EXT) is what controls whether or not the library command's external calls will be extracted as well; this is needed if you're trying to make a subset of a library.

The second SETPREF option (DIR) is what controls whether or not a function's header will be extracted as well; this is not normally desired.

'$PRG' (when present) is the actual code of the extracted function. Otherwise, its code will be in a variable named the same thing as the command being extracted.

To make a sublibrary (a library which is a subset of a larger library) run SETPREF, set EXT to ON and DIR to OFF, place a list of the desired commands on the stack, and run LIBEX. Now use the <-LIB-> library to create a $ROMID, $TITLE, and then run MCFG and D->LIB.

STR33

Author: Todd Eckrich (mte@delphi.com)
[Note: Todd's program actually was STR22; I modified it to STR33 for use with Jazz's ED. -jkh-]

This small program formats text to fit within the 33-character wide display of Jazz's ED. The way it does it, however, is different from other similar programs. A machine code routine simply rearranges the space and linefeed characters in two passes. The first pass simply replaces all linefeeds with spaces. The second pass puts a linefeed at the first space encountered backwards from the 34th character of each line. As a result, words do not get haphazardly split and the program is extremely fast. If there are more than 33 consecutive nonspaces, then the 34th character is replaced with a linefeed. The argument is a character string. It does not work for formatting source code. It is intended more for text, especially editing large text files.

==========

ED33
A Fast Object Editor

by Joseph K. Horn

Instead of pressing down-arrow (or EDIT) to edit an object, run ED33. It formats the object just like the built-in editor, except with margins 33 wide (instead of the usual 19), and then uses Jazz's ED to edit it.

Not for use with System RPL and/or Code objects; use DIS/ED/ASS for those. ED33 is primarily intended for editing quick-n-dirty User RPL programs.

```
by Mika Heiskanen.
Offered as-is, strictly for adventuresome hackers.

Name:   COERCE
Desc:   Object conversions
Stack:  %       --> #
        hxs     --> #
        #       --> %
        %%      --> %
        C%%     --> C%
        chr     --> $
        TRUE    --> %1
        FALSE   --> %0

Name:   XRCL
Desc:   RCL replacement
Stack:  $pk     --> ob          Calls UPK
        id      --> ob
        lam     --> ob
        PICT    --> grob
        {seco}  --> seco
        {romptr}--> ob
        {path}  --> ob
        {}      --> ob
        romptr  --> ob
        hxs_addr--> ob
        #addr   --> ob
        acptr   --> ob
        %port   --> Pvars
        %lid    --> Libs        With port numbers
        :&:lid  --> lib         Works for built-in libs too
        tagged  --> ob

Name:   STO2
Desc:   STO replacement
        Libraries will be immediately in use, a possible configuration
        object will be replaced by a TOSRRP call. The code will report
        if it suspects TOSRRP is not enough, in which case ON-C should
        be done to full execute the configuration object. Lid 4 is
        reserved for a crash library and no attach will be done.
Stack:  ( ob tag --> )
        ( ob id  --> )
        ( ob lam --> )
        ( ob symb --> )
        ( grob pict --> )
        ( backup %port --> )
        ( lib %port --> )

Name:   TIM
Desc:   Measure execution time in milli seconds
Stack:  ob --> ? %msecs         Not accurate on S

Name:   USEND
Desc:   Send object via IR
Stack:  ?

Name:   URECV
Desc:   Receive object from IR
Stack:  ?

Name:   BZ
Desc:   Compressor / Uncompressor. Very fast, efficient for large
        objects.
Stack:  $bz --> ob | ob --> $bz
```

```
Name:   RFU
Desc:   Uncompress RF'd object
Stack:  $rf --> ob

Name:   SYS
Desc:   SYSEVAL + some conversions
Stack:  hxs_addr--> ?           SYSEVAL
        %       --> %%
        C%      --> C%%
        %%      --> %
        C%%     --> C%

Name:   COD
Desc:   Convert hex chars to an object. Ignores whitespace.
Stack:  $ --> ob

Name:   DCOD
Desc:   Convert object to hex dump. Shows rpl structure
        [For a pure hex conversion use ->ASC.  -jkh-]
Stack:  ob --> $

Name:   OBJFIX
Desc:   Fix bad download
Stack:  $bad --> ob

Name:   VARS
Desc:   ML VARS replacement
Stack:  --> {ids}

Name:   VARS2
Desc:   SRPL VARS replacement, list nullids too
Stack:  --> {ids}

Name:   PG
Desc:   PURGE replacement
Stack:  tagged -->
        id -->
        {ids|tags} -->
        PICT -->

Name:   PG0
Desc:   Purge port0 (Calls PG on 0 PVARS)
Stack:  -->

Name:   ORD
Desc:   ML ORDER replacement
Stack:  {ids} -->

Name:   REN
Desc:   Rename variable
        [Caution! Do not use in very-low-memory situations! -jkh-]
Stack:  id_new id_old -->

Name:   TB
Desc:   Tabify srpl/ml source code.
Stack:  $ --> $'

Name:   FMT
Desc:   Convert data strings into readable format. size determines the
        nibble count of each word, vars how many shall be put on each
        line
Stack:  $ %size %vars

Name:   ITYPE
Desc:   Get internal type number of object (as used by Dispatch)
Stack:  ob --> #type
```

```
Name:   CTIM
Desc:   Measure cycle count for instruction relative to P= 0
        instruction by making a test program with Jazz assembler.
        Example:  "A=DAT1 A" --> %13.801         (GX rev P)
Stack:  $ --> %cycles

Name:   DTEMP
Desc:   Dump non-bints from tempob area, as many as possible without GC
Stack:  --> obs

Name:   MEM1
Desc:   Get free memory for card1, GX only
Stack:  --> %bytes

Name:   CDHD
Desc:   Set context properly to hidden directory

Name:   WKEY
Desc:   Get key object for next full key press
Stack:  ( --> ob %keycode )

Name:   SC
Desc:   Memory scanner by Rick Grevelle

Name:   ->ASC
Desc:   Convert object to hex string. [To see internal structure of
        RPL objects, use DCOD instead. -jkh-]

Name:   USE
Desc:   Report objects usage of ids and romptrs.
Stack:  ( ob --> {} )
        ( rrp --> { name1 {} name2 {} .. )

Name:   PMEM
Desc:   Return free memory in port
Stack:  ( %port --> %bytes )

Name:   D->LIB  [Code by Rick Grevelle.  -jkh-]
Desc:   Create library
Stack:  ( rrp --> lib )
        (     --> lib )         CONTEXT used

Name:   L->DIR  [Code by Rick Grevelle.  -jkh-]
Desc:   Split library. Notes:
                - $CONFIG is not converted
                - Cannot handle internal HP libraries
Stack:  ( lib --> rrp )
        ( #lid | %lid | hxs_lid --> rrp )

Name:   OB->
Desc:   Split object
Stack:  ( arry --> obs {dims } )
        ( seco --> obs %n )
        ( symb --> obs %n )
        ( romp --> %lid %cmd )
        ( rrp --> obs %n )
        ( backup --> ob id )
        ( id --> $ )
        ( #addr --> ob )
        ( acptr --> ob )
        + built-in stuff

Name:   ->DIR
Name:   ->PRG
Name:   ->XLIB
Name:   ->ALG
Name:   ->BAK
Name:   ->ID
```

```
Name:   ADDR
Name:   LBCRC  ( lib --> repaired_lib )
Name:   RHASH  ( #lid | %lid | hxs_lid --> hash_table )
Name:   RLINK  ( #lid | %lid | hxs_lid --> link_table )
Name:   RCFG  ( #lid | %lid | hxs_lid --> config )
Name:   RMSG  ( #lid | %lid | hxs_lid --> message_table )
Name:   RTITLE  ( #lid | %lid | hxs_lid --> $title )

Name:   XGET
Desc:   XRECV substitute to enable downloading big objects. (FXRECV)
Stack:  ( id | %port --> )

Name:   BZD
Desc:   Pack directory.
Stack:  ( --> )
        [Note: This runs BZ on *every* object in the current directory
        and replaces the original objects with their BZ'd counterparts
        without warning. -jkh-]

Name:   USES
Desc:   Reports which vars in a directory call an id or romptr.
Stack:  ( rrp id | romptr --> { } )

Name:   USED
Desc:   Report var refs for an entire dir.
Stack:  ( rrp --> { } )
        [Note: the output is a list of the var names in the directory
        with each being followed by a list of all the vars which
        reference it.  This is like the inverse of USE.  -jkh-]

Name:   GRX2
Desc:   GROB expand 2X
Stack:  ( grob --> bigger_grob )
        [Note: DON'T USE THIS!  It is buggy, and crashes often.
         Fixing it is left as an exercise for the student.  -jkh-]

Name:   BY
Desc:   Byte size of any object
Stack:  ( any --> %size )
        [Note: 2.5-byte ROM objects are copied to TEMPOB first.
        -jkh-]
```

==========

SPORT (Search PORT)
Fast GX Port Searcher by Dave Marsh
Modified by Joe Horn

INPUT: Name or Library ID (GX ONLY!!!)
OUTPUT: List of all the ports in which that name or library resides.

Typical runtime: 0.2 seconds.

Note: Works with any configuration of RAM and/or ROM cards, up to 128K in port 1 and 1 Meg in port 2.

==========

VV33
A Fast Object Viewer

by Joseph K. Horn

Like VV, except VV33 works with any kind of object.

Not for use with System RPL and/or Code objects; use DIS/VV for those. VV33 is primarily intended for viewing User RPL programs.

MKCOPY
Make 'COPY' in RAM

by Rick Grevelle and Joseph K. Horn.

MKCOPY creates a powerful ROM-copier program and stores it into a variable called 'COPY' in your current directory. MKCOPY is safe, but COPY is very dangerous. Read the following VERY carefully.

Documentation for the 'COPY' program:

GX ONLY! Intended ONLY for those who OWN an application ROM card AND a port 2 RAM card (e.g. HP 1-Meg RAM card), and wish to copy the ROM card into the RAM card so that port 1 is free for other uses.

> NOTICE!!! This is NOT intended for use by software pirates and other scurvy marauders. COPYing ROM cards that you do not own (or making copies of your ROM cards for other people) is illegal, immoral, and stupid. It's illegal because it hurts commerce. It's immoral because it denies a worker his just wages. And it's stupid because it discourages good programmers from writing better programs. DON'T.

WARNING! This is a "bit copier", that is, it makes an EXACT copy of the card which is in port 1. This can be excellent, or nightmarish. The previous contents of the target port will be totally overwritten and irretrievably lost. If port 1 contains code that cannot run in port 2 or above, COPY will copy it anyway, which can cause a crash if you turn your HP48 back on without removing the new copy.

KNOW WHAT YOU'RE DOING. WHEN IN DOUBT, DON'T. YOU HAVE BEEN WARNED!

INSTRUCTIONS:

First, create the COPY program by running MKCOPY in the HP48 Hacker's ROM, if you haven't already done so. Then turn off your HP48 and remove the Hacker's ROM. Plug the desired ROM card into port 1, and make sure a RAM card is in port 2.

Turn the HP48 back on. Input the PORT number (2 through 33) which you wish to RECEIVE the copy. Run COPY. When the copying is finished, the HP48 will turn itself off. DO NOT TURN BACK ON until at least one of the cards is removed. If the ROM card that was copied is a card that must be run from port 1 (such as the Hacker's ROM) then be sure to remove the copy from port 2 before turning the HP48 back on.

Possible Error Message:

Port Not Available (either port 1 contains no card, or the target port doesn't exist).

Note: if the copy causes the HP48 to refuse to turn on (for example, a library's configuration routine might go into an endless loop), DON'T PANIC. You need not lose the entire contents of your RAM card in slot 2. All you have to lose is the offending libraries which COPY copied from port 1 to your RAM card. This can be done by removing the RAM card from slot 2 and then copying the XPUB library (on GD10) into RAM. XPUB prevents libraries from running their configuration routines. Purge the offending libraries, and then purge XPUB.

Disclaimer: This is dangerous software. Use at own risk.

Code

'Code', a program that converts any HP48 object into a Code object, and back again. By "Ram" Gudavelli, Richard Steventon, & Joe Horn.

Input: any

Output: Code (if input was non-Code)
    or: obj (of input was Code)
    or: "Undefined Result" error (if Code was not created by 'Code')

There is a small outside chance that 'Code' will mistakenly think that a Code object was created by 'Code' which in fact was not, but the probability is extremely low, about 30 in a million. I've been unable to date to make it crash. It's still a good idea to backup memory before running it willy-nilly on Code objects that 'Code' didn't create.

The ->Code logic is by ram.gudavelli@nybble.com.
The Code-> logic is by Richard Steventon (lstevent@cs.uct.ac.za).
The safeguards & auto-selection were added by Joe Horn.

Suggestion: You can greatly speed up your HP48's keyboard response in USER mode by making sure that each of your assignments is a single object. If you have any program objects assigned to keys, use Code to convert them to Code objects, and reassign them.

FBROW
Fast Flag Browswer

by Richard Steventon

Features:
==========
- 10 minute timeout to save batteries.
- Very fast (this is important ;)
- Postcard-ware (more about this later).

Keys:
=====
- Up and Down Arrow keys to scroll up and down
- [NXT] changes between user and system flags
- [+/-] toggles the flag highlighted by the scroll bar
- RS + Up/DnArrow keys = Top/Bottom of flag list
- [ENTER] saves flag changes and exits
- [ATTN] exit and don't save changes
- [H] Help/info screen (with cute GROB)
- [0] to [6] jump to that flag base eg [2] sets flag 21 to the top.
- Other keys result in a "DoBadKey" beep.

Postcard-ware:
==============
Postcard-ware means that you *should* (ie please) send me a postcard to say thanks if you find FlagBrowser v3.0 useful. If I get a lot of postcards (more than 1), maybe people (my mother) will believe me when I tell them I am working (on my HP) and not "playing". This will mean that I will have more time to produce other things for the hp48 ! So, here is my address again in case you can't bear to press the [h] key in FlagBrowser:

Richard Steventon
7 Sun Valley Avenue
Constantia, 7800
South Africa

Credits:
========
Joe Horn for his patience and extensive testing.
Mika for Jazz, DB (without which, I could not have done FlagBrowser),
and his kind permission to use his keyhandler from ED.
Mozgy for permission to include FNT1 in the binary.
Dan Kirkland for the phrase "Mine WILL be better!"
Jon Paine for lending me his GX and organising a cheap one for me.
Bill Wickes and Raymond Hellstern for their flag browsers.
My girlfriend, who put up with me while I "played" for many hours.
Everybody on IRC channel #hp48 for comments, encouragement, etc

That is about it...

If you have any questions/comments/flames/lawsuits/etc then feel free to email me.

-Richard Steventon

ps, to contact me: <email> lstevent@cs.uct.ac.za (until 28 Feb 1996)
IRC RichardS on channel #hp48 (normally 1pm GMT)

UPD
UPDIR done right!

by Joseph K. Horn

Assign << 1:UPD EVAL >> to your UP (UPDIR) key. Better yet, assign this:

```
::
 Dolst/2nd+:
 xUPDIR
 TAG 1
 ID UPD
 xEVAL
;
```

This will solve a major headache of HP48 usage, especially for programmers who have many variables in RAM. We've all experienced this: every time you press the UP key to execute UPDIR, it resets the menu to page 1, and so you always have to press NXT, NXT, NXT... to get back to the menu page that you just came up from. I always used to press NXT too many times, and have to then press PREV to back up a page or two. It's a waste of time. It's aggravating. And now it's unnecessary.

With 'UPD' assigned to your UP key, and you'll love the way it does UPDIR, automatically jumping to the menu page containing the directory that you just came up from. And it's VERY fast. You'll never have to wade through your VAR menu again!

Hacker's note: UP will act the same as UPDIR if you run it from a hidden directory.

==========

BZM
BZ Menuline

by Jack Levy

For those of you that have used BZ and wanted to create string or program
compressions, here is a menuline that will allow you to do so. Once you execute BZM, a menu appears with four options:

BZ - Execute BZ compressor on the level1 argument.

SFX (Self-Extracting) - Intended for compression of strings, lists, or other non-program objects. First executes BZ on the level1 argument, then attaches a BZ to the end of the compression string. Thus, when the result is stored in a variable, when you press the variable key, the string will come up without the need for you to run UBZ.

EXE (Self-Extracting-and-Executing) - Intended for compression of programs. Executes BZ on the level1 argument, then attaches a UBZ EVAL at the end of the string. When the result is stored in a variable, executing this variable will result in your program being run with no need for manual decompression.

UBZ (Un-BZ) - Unpacks the level1 BZ'd argument. If the argument is a SFX or EXE created argument (see above), it will extract the compressed string and execute UBZ as normal.

Caution: Do not use UBZ on programs that were not created by SFX or EXE, or undesired results will occur.

HTRIM
HOME directory Trimmer

by Simone Rapisarda

HTRIM is a sys-RPL program that I use to hide variables in the HOME directory so to keep it a clean and tidy (yes, you've read well: HOME directory, not Hidden directory!).

HTRIM is also a command in the SmartKeys library. You can use HTRIM in other libraries or programs only if these are Public Domain or Freeware. Anyway this should be done only with my consent and their use and origin must be reported in the documentation of the software.

Here is what the SmartKeys manual says about HTRIM:

```
+-------+
| HTRIM |           (Homedir-TRIM)
+-------+
```

Stack: { id id id ... } or %0

It works only in the HOME directory.

If the argument is a List HTRIM orders the variables specified in the List as the built-in command ORDER does, the only difference is that all the remaining variables becames hidden: you can still use them as usual but they won't appear on the VAR menu and in the List created by the VARS command. To hide all the variables use an empty List. If the argument is the Real 0, HTRIM brings back to light all the hidden variables. This command is very useful to keep clean and tidy the VAR menu of the HOME directory.