

CL-UTILS ROM

41CL Extensions Module.

User's Manual and Quick Reference Guide



©Photo By Jürgen Keller, 2011.

Programmed by Ángel M. Martín

December 2011

This compilation revision 2.F.02

Copyright © 2011 Ángel Martín

Published under the *GNU* software licence agreement.

Original authors retain all copyrights, and should be mentioned in writing by any part utilizing this material. No commercial usage of any kind is allowed.

Screen captures taken from V41, Windows-based emulator developed by Warren Furlow.
See www.hp41.org

CLWRITE Source Code written by Raymond Wiker.

Cover photo © Juergen Keller, 2011.
Inside photos © Geoff Quickfall, 2011

Acknowledgment.- This manual and the CLUTILS module would obviously not exist without the 41CL. Many thanks to Monte Dalrymple for the development of the amazing CL board.

CL-UTILS Module

Extension Functions for the 41CL

Table of Contents.

1. Introduction.	
1.1. A word of Caution.	5
1.2. The Functions at a glance	6
2. The functions in detail	
2.1 Function Launchers	7
2.2 Catalogues and CATalogs	9
2.3 Interrogating the MMU	12
2.4 A wealth of a Library	13
3. HEPAX and Security	
3.1. Configuring the HEPAX system	14
3.2. Security Functions	16
4. Advanced Territory	
4.1. Using Page #4	17
4.2. Calculator Flash Backup & Restore	17
5. Other Extensions	
5.1 Alpha and Display Utilities	19
5.2 Other Utilities	20
5.3 Farewell.	
6. Appendixes.	21

CL-UTILS Module

Extension Functions for the 41CL

1. Introduction.

Without a doubt the 41CL can be considered in many ways to be the pinnacle of the HP-41 system. It comes with a well thought-out function set to manage its capabilities, from the basic to the more adventurous ones – which have inspired the writing of yet further extensions to that capable toolset.

This collection is designed to enhance and complement the YFNS function set, providing easier access to the many powerful capabilities of the 41CL platform. Some are function launchers, grouping several functions by their area of functionality into a single, prompt-driven one – like it's the case for the Plug/Unplug functions, the Baud rate, TURBO and MMU settings functions. A launcher of launcher sits atop these, providing quick access to 27 YFNS functions from a single key assignment.

Some other extend the functionality by providing new features and more convenient alternative to manual tasks. Examples of these are:

- A fully-featured ROM library CATALOG system, allowing direct plugging into the port of choice
- The *Page* Plug functions (alternative to the *Port* ones), including routines to handle page #4.
- Programs to backup and restore the complete calculator contents to/from Flash
- HEPAX configuration and set-up, making the HEPAX integration a simple and reliable affair.
- Security functions to password-protect your machine from prying hands.

Other housekeeping functions roundup the set, making for a total of 41 functions tightly packed into a 4k ROM. This is a design criterion, as the small footprint of the module makes it ideal to share with other utility packs, most notoriously the CCD OS/X (or its alter-ego AMC OS/X) for the ultimate control - so save some small exceptions there is no duplication between these two.

A word of caution.

As wise men remind us all, "*with power comes responsibility*". Indiscriminate usage of some of these functions can have unpleasant consequences, ranging from unexpected results and easy-to-recover machine lock-ups to more serious ones involving loss of Flash sectors or even electrical damage in the worst scenario. Functions have some built-in protection to ensure that they're used properly, but they are not absolutely foolproof in that such protection can always be circumvented. So beware, and as general rule "*if you don't understand something, don't use it*".

To help you with this the more dangerous functions are marked with the **WARNING** sign all throughout this manual. Avoid them if you're not absolutely sure that you know what they are for, and fully understand their operation. And *always, always have fresh batteries on when using the Flash backup!*

It had to be said – so now that we got it out of the way we're ready to dive into the CL UTILS description and usage example. May you have a nice ride!

Function index at a glance.

And without further ado, here's the list of functions included in the module:

#	Function	Description	Inputs	Output
1	-CLUTLS 2E	<i>Module Header</i>	<i>n/a</i>	<i>n/a</i>
2	?MMU	MMU Status Yes/No	None	YES/NO, skip if false
3	ΣCLF _	Global Launcher	Prompts "B:M:P:T:U"	Launches selected Launcher
4	BAUD _	Baud functions launcher	Prompts "1:2:4:9"	Launches selected function
5	CLLIB _	CL ID Library	Prompts "A-Z"	Starts listing at selected letter
6	MMU _	MMU functions launcher	Prompts "C:D:E:?"	Launches selected function
7	MMUCAT	MMU Catalogue	None	Sequential list of MMU Entries
8	PLUGG _	Plug Page	Prompts for page	Plugs ROM in page
9	PLUGG? _	Page Location MMU	Prompts for page	content of MMU entry for page
10	PLUGGX	Plug Page by X	Page# in X	Plugs ROM in page
11	PLG#4 _	Page#4 Plug	Prompts "F:L:S"	Selected ROM plugged
12	PLUG _	PLUG functions launcher	Prompts for location	ROM with ID in ALPHA is plugged
13	ROMLIB	ROM Library	Displays all ROMs	Sequential list of ROM ID's
14	UPG#4	Clears MMU entry for page #4	None	MMU entry cleared
15	TURBO _	TURBO functions launcher	Prompts "X:2:5:1:0:,:?"	Launches selected function
16	UPLUG _	UPLUG functions Launcher	Prompts for location	Location is removed from MMU
17	Y1SEC	One-second delay	None	1-sec delay
18	YBSP	ALPHA back Space	String in ALPHA	Deletes rightmost character
19	YCL>	Clears string from ">"	String in ALPHA	Clears from ">" char to the right
20	YCL-	Clears string from hyphen	String in ALPHA	Clears from "-" char to the right
21	YFNZ?	Page location of YNFS	None	Location in MMU
22	YINPT _	Y Input	None	HEX entry plus control chrs.
23	YRALL	Y-Read-ALL	None	Reads Calculator/MMU from Flash
24	YSWAP>	Swaps both sides of ">"	String in ALPHA	Alpha swapped around ">"
25	YSWAP-	Swaps both sides of hyphen	String in ALPHA	Alpha swapped around "-"
26	YWALL	Y-Write-ALL	None	Writes Calculator/MMU to Flash
27	-SYS/EXT	<i>Section Header</i>	<i>n/a</i>	<i>n/a</i>
28	ADRID	Address ID	Flash address in Alpha	ROM ID in Alpha
29	BFCAT	Buffer Catalogue	None	Shows present buffers
30	BLCAT	Block Catalogue	None	Lists block contents
31	CDE	Code	HexCode in ALPHA	NNN in X
32	DCD	Decode	NNN in X	Hex Code in Alpha
33	DTOA	Display to ALPHA	Display contents	Text in Alpha
34	DTST	Display Test	None	Shows display all lit up
35	HEPINI	HEPAX FileSys Init	# pages in X, first page in Y	Initializes HEPAX File System
36	"HPX4"	HEPAX FileSys Init - CL	None	Configures 4k HEPAX on CL
37	"HPX8"	HEPAX FileSys Init - CL	None	Configures 8k HEPAX on CL
38	"HPX16"	HEPAX FileSys Init - CL	None	Configures 16k HEPAX on CL
39	SECURE	Enable password lock	None	Sets SECURE mode ON
40	UNLOCK	Disable password lock	Asks for password	Sets Secure mode OFF
41	XPASS	Change password	Asks old/new passwords	Password is changed

Functions in **BLUE** are all in MCODE.


Functions in **BLACK** are MCODE entries that call FOCAL programs.

Functions in **"QUOTES" italics** are FOCAL programs.

Functions in **RED** denote prompting entries.

2. The functions in detail.

The following sections of this document describe the usage and utilization of the functions included in the CL-UTILS module. While some are very intuitive to use, others require a little elaboration as to their input parameters or control options, which should be covered here.

2.1- LAUNCHERS	
----------------	--

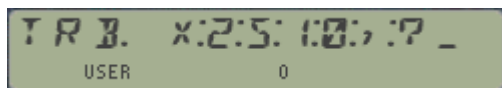
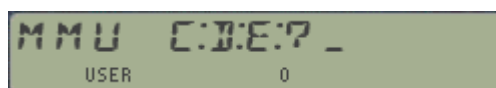
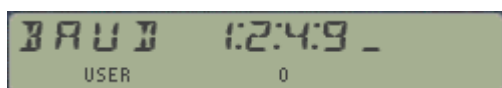
2.1 Function Launchers.

The table below lists the launchers by function groups:

Index	Function	Warnings	Description
1	BAUD	None	Calls BAUD12, BAUD24, BAUD48, or BAUD96
2	MMU	None	Calls MMUDIS, MMUEN, or MMU?
3	TURBO	None	Calls the corresponding TURBO _{xx} function
4	PLUG	Light	Prompts for port location. Enter L/U first (when needed)
5	UPLUG	Light	Prompts for port location. Enter L/U first (when needed)
6	ΣCLF	None	Launcher of Launchers -> invokes any of the five above

When you assign ΣCLF to any key that alone will give you access to more that 25 functions from that single key – an effective way to make it compatible with other existing key-assignments, saving memory (KA registers) and time. So go ahead and get comfortable with that arrangement as your baseline.

Prompting functions use a technique called partial key entry, dividing the data entry in two (or more) parts. The keyboard is also re-defined, in that just those keys corresponding to the appropriate options are active. The cues in the prompt will offer you indication of which keys are active on the keyboard, and typically are intuitive enough to figure out in each case.



options for: none, 2x, 5x, 10x, 20x, 50x, and ?

Use "0" for 20x, Radix for "50" - as 2 and 5 are already taken for 2x and 5x speeds.

In general all launchers behave in a similar manner.

- The Back Arrow key will either cancel out entirely or remove partial entries;
- Non-active keys will blink the display and maintain the prompt
- Holding down the last key briefly shows the invoked function name – visual feedback.
- This will be followed by NULL if kept depressed long enough – last chance to bail out.
- Launchers are not programmable per-se – but:
- They can be used in PRGM mode to enter the called-upon function as a program line.

The **PLUG** and **UPLUG** launchers don't offer any cues in the prompt – and therefore deserve special consideration. The picture below shows the convention for the external pages of the 41:

Port 1 <i>Upper Page (9-hex)</i> <hr/> <i>Lower Page (8-hex)</i>	Port 2 <i>Upper Page (B-hex)</i> <hr/> <i>Lower Page (A-hex)</i>
Port 3 <i>Upper Page (D-hex)</i> <hr/> <i>Lower Page (C-hex)</i>	Port 4 <i>Upper Page (F-hex)</i> <hr/> <i>Lower Page (E-hex)</i>

Valid entries for the prompt are:

- 1, for port 1 – comprising pages 8 and 9
- 2, for port 2 – comprising pages A and B
- 3, for port 3 – comprising pages C and C
- [L], to flag a LOWER half-port condition, followed by the port number
- [U], to flag an UPPER half-port condition, followed by the port number



For the **(U)PLUG** cases the prompt completes either when the number 1-4 or the letter {G,P,H} is entered, and the corresponding function is launched.

For half-port (or 4k) modules *use the L/U keys first* in the (un)plugging prompts, then the port number. These keys act as toggles when pressed sequentially, replacing each other in the display upon repeat usage. Also during these events pressing BackArrow removes the half-port condition and returns to the main prompt.

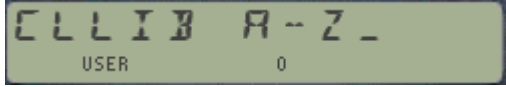
Remember that plugging a module into the “wrong” port location can create minor issues (or major havoc) if you're overwriting some/part of the machine's configuration. A good example is overwriting YFNS itself, or a HEPAX RAM block. Always make sure the destination is safe – using BLCAT, the standard CAT2 or better yet the CCD CAT'2.

Also valid entries are :

- [H], for page #7 – the HP-IL reserved page
- [P], for page #6 -- the Printer reserved page
- [G], for page prompt 6-F – effectively calling the **PLUGG(?)** functions

Caution.-

Both PLUG/UNPLUG offer all the 14 available choices in **YFNS**, including **(U)PLUGP** and **(U)PLUGH**. Exercise extra caution with those two locations, as they may be used by system extensions like Printer or HP-IL. Page #6 in particular has more strict demands on the ROM layout that makes it non-suitable for the majority of ROMs. Also because pages #6 and #7 on the CL don't support bank-switching, they unfortunately aren't a good place for the HEPAX ROM.

2.2.- CATALOGS	
----------------	--

2.2. CATALOGS, CATALOGUES...

The additional CATalogs are as follows:

Index	Function	Warnings	Description
1	BLCAT	None	Borrowed from the HEPAX ROM – shows the 4k-blocks contents.
2	BFCAT	Light	Lists those elusive buffers present in the system.
3	MMUCAT	None	Lists the MMU mappings into each block.
4	ROMLIB	Light	List the ROM Library ID's available in Flash.
5	CLLIB_	Light	Same as above with an Alpha prompt for beginning section

If you're like me you'll like to have good visibility into your machine's configuration. With its ROM Library and MMU settings the CL adds a few dimensions to the already rich 41CX system – and the goal is to have equivalent catalogue functions to review the status and options available.

Each CATalog has its own idiosyncrasies, but in general they feature single-step modes, and have “hot keys” to allow for specific actions – like deletion of buffer, navigation shortcuts, and direct plugging of ROMs into a port. This makes chores like searching for the correct syntax and plugging a module from the library a trivial task.

Both **BLCAT** and **BFCAT** are not strictly related to the CL, and will also work on a standard 41. Obviously **MMUCAT** is only meaningful for a CL machine, and will return all zeroes if the CL board is not installed.

CATalog functions are notoriously complex and take up a significant amount of space – yet you'd hopefully agree with me that the usability enhancements they provide make them worthwhile the admission price.

2.2.1. Block CATALOG

BLCAT	Block Catalog	Author: VM Electronics	Source: HEPAX Module
--------------	----------------------	------------------------	----------------------

Lists the first function of every non-empty ROM block (i.e. Page), starting with Page 3 in the 41 CX or Page 5 in the other models (C/CV). The listing will be printed if a printer is connected and user flag 15 is enabled.

- Non-empty pages will show the first function in the FAT, or **“NO FAT”** if such is the case
- Empty pages will show the **“NO ROM”** message next to their number.
- Blank RAM pages will show **“QUASI RAM”**, indicating their RAM in ROM space character.

No input values are necessary. This function doesn't have a “manual mode” (using **R/S**) but the displaying sequence will be halted while any key (other than **R/S** or **ON**) is being depressed, resuming its normal speed when it's released again.

2.2.2. Buffer CATALOG

BFCAT	Buffer CATalog	Hot keys: R/S, SST, SHIFT, D, H	
[D]	Deletes Buffer	<i>In manual mode</i>	
[H]	Decodes Header register	<i>In manual mode</i>	

This function is very close to my heart, both because it was a bear to put together and because the final result is very useful and informative. It doesn't require any input parameter, and runs sequentially through all buffers present in the calculator, providing information with buffer id# and its size.

41 buffers are an elusive construct that is mainly used for I/O purposes. Some modules reserve a memory area right above the KA registers for their own use, not part of the data registers or program memory either. The OS will recognize those buffers and allow them to exist and be managed by the "owner" module – which is responsible to claim for it every time the calculator is switched on.

A good example is the Time module, which uses it to store the alarms data.

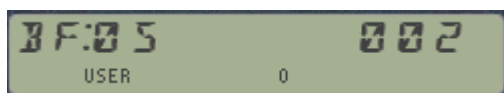
Each buffer has an id# number, ranging from 1 to 14. Only one buffer with a given id# can exist, thus the maximum number present at a given time is 14 buffers – assuming such hoarding modules would exit – which thankfully they don't.

The table below lists the well-known buffers that are possibly to be found on the system:

Buffer id#	Module/Eprom	Reason
1	David Assembler	MCODE Labels already existing
2	David Assembler	MCODE Labels referred to
3	Eramco RSU-1B	ASCII file pointers
4	Eramco RSU-1A	Data File Pointers
5	CCD Module, Advantage	Seed, Word Size, Matrix Name
6	Extended IL (Skwid)	Accessory ID of current device
7	Extended IL (Skwid)	Print Cols, number & width
8	Complex Stack	Ángel Martin's 41Z ROM
10	Time Module	Alarms information
11	Plotter Module	Data and barcode parameters
12	IL Development, CMT-200	IL buffer and monitoring
13	CMT-300	Status Info
14	Advantage	INTEG & SOLVE scratch
15*	Mainframe	Key Assignments

*) KA area isn't really a buffer.

For instance, plug the AOSX module into any available port. Then type **PI**, **SEED**, followed by **BFCAT** to see that a 2-register buffer now exists in the 41 I/O area – created by the **SEED** function.



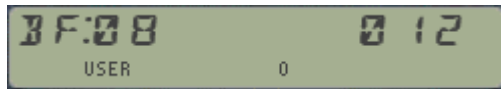
id# = 5, buffer size = 2, properly allocated.

Suppose you also change the default word size to 12 bits, by typing: 12, **WSIZE**. This has the effect of increasing the buffer size in one more register, thus repeating **BFCAT** will show:



id# = 5, buffer size = 3, properly allocated.

Say now that you also plug the 41Z module into a full port of your CL. Just doing that won't create the buffer, but switching the calculator OFF and ON will – or alternatively execute the **-HP 41Z** function. After doing that execute **BFCAT** again, then immediately hit **R/S** to stop the listing of the buffers and move your way up and down the list using **SST** and **BST**. You should also see the line for the 41Z buffer, as follows:



id#=8, buffer size = 12, properly allocated.

If the module is not present during the CALC_ON event (that's to say it won't re-brand the buffer id#) the 41 OS will mark the buffer space as "reclaimable", which will occur at the moment that PACKING or PACK is performed. So it's possible to have temporary "orphan" buffers, which will show a question mark next to the id# in the display. This is a rather strange occurrence, so most likely won't be shown – but it's there just in case.

BFCAT has a few hot keys to perform the following actions in manual mode:

1. **R/S** stops the automated listing and toggles it with the manual mode upon repeat pressings.
2. **[D]** – for instant buffer deletion – there's *no way back, so handle with care!*
3. **[H]** - to decode the buffer header register. Its structure contains the buffer ID#, as well as some other relevant information in the specific fields - all buffer dependent.
4. **[SHIFT]** to flag the listing to go backwards – both in manual and auto modes.
5. **SST** and **BST** to move the listing in manual mode, until the end (or beginning) is reached
6. **BackArrow** to cancel out the process and return to the OS.

Like it is the case with the standard Catalogues, the buffer listing in Auto mode will terminate automatically when the last buffer (or first if running backwards) has been shown. In manual mode the last/first entry will remain shown until you press BackArrow or R/S.

Should no buffers are present, the message **"NO BUFFERS"** will be shown and the catalog will terminate. Note also that the catalogue will not be printed - being shown only on the display.

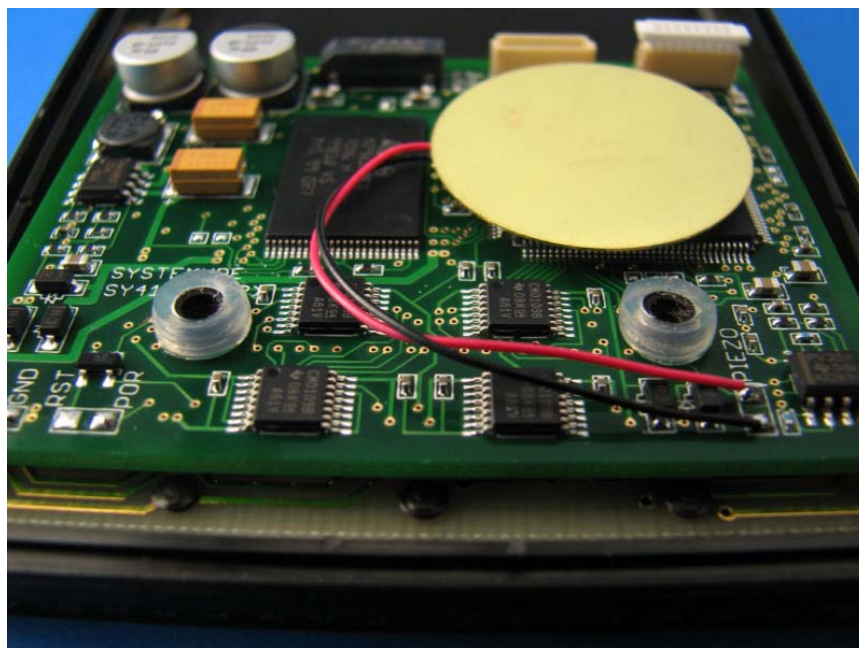


Photo courtesy of Geoff Quickfall.

2.2.3. Interrogating the MMU.

MMUCAT	MMU CATalogue	No inputs	
ADRID	Gives ROM id# from ADR	Expects string in Alpha	
FYNZ?	FYNS Location Finder	No inputs	
PLUGG?	ROM id# in page by X	Prompts for page#	Valid inputs are 4, 6-F

MMUCAT is really a FOCAL program that drives the function **ADRID**, the real engine behind it – not to be confused with the capital city of a country I know quite well. **ADRID** is obviously programmable. The idea is simple: produce a list of the MMU mappings into the different pages, showing either the ROM id# or the address (Flash or SRAM) currently mapped to the port.

A loop is executed starting on page #4, and up until page #F. Each iteration retrieves (pokes more appropriately) the address written into the corresponding MMU register, then searches it against the internal ROM id# table written into the CL_UTILS module. More about this later.

Note that full-port modules will return the ROM id# attached to the lower half, and the address to the upper half. RAM MMU entries will return the corresponding RAM address.

While similar to the CAT2 concept, this really has an MMU-oriented perspective of things, and thus is purely a 41 CL feature – it'll render all entries zero if used on a "regular" 41. The program listing is rather simple – as **ADRID** does all the weight lifting under the hood:

01	LBL "MMUCAT"		20	LBL 00	
02	"0-0000"		21	"8040"	prefix
03	ASTO X		22	XTOA	
04	52	"4"	23	ARCL Y	page#
05	XEQ 00		24	YPEEK	read MMU rg,
06	CLX		25	YSWAP-	swap around "-."
07*	54,057	"6" to "9"	26	YCL-	delete from "-."
08	LBL 02		27	YBSP	back space
09	XEQ 00		28	ATOX	
10	ISG X	next	29	RDN	
11	GTO 02		30	ADRID	decode address
12	CLX		31	XTOA	
13*	65,07	"A" to "F"	32	" :-"	
14	LBL 01		33	-3	
15	XEQ 00		34	AROT	
16	ISG X	next	35	RDN	
17	GTO 01		36	AVIEW	show ID#
18	CLD		37	PSE	pause
19*	RTN		38	END	

A related function is **FYNZ?**, which returns the page number the YFNS is currently plugged in. This can come very handy in your programs to avoid overwriting it with other modules – as we'll see in the HEPAX configuration routines.

Another related function is **PLUGG?** - It interrogates the MMU to find out which module is plugged into a given page – the input to the function placed in X. This is all **page**-driven, and not based on the *port* number. There is no restriction in the input to the page number, however the returned values for pages 0,1,2,3, and 5 don't quite have the same meaning.

PLUGG? Also uses **ADRID** to decode the string returned by **YPEEK** – which provides the MMU address mapping the corresponding page. In the **FYNZ?** case there's no need to look up in the ROM id# table since we know what we're looking for – just need to check all pages looking for that specific string.

2.2.4. A wealth of a Library.

ROMLIB	ROM Library	No inputs	
CLLIB	CL Library	Prompts for A-Z	
[P]	Invokes PLUG _		
[A]	Copies id# shown to Alpha		

One of the most notable features of the CL is its extensive ROM image library, allowing you to plug almost any conceivable module ever made (of which I have contributed a few) into your 41CL just by using one of the **PLUGxx** functions. The input syntax requires that the correct ROM ID string be placed in Alpha, and certainly there are a few of those to remember – and rather similar to each other since the string is only 4 characters long.

These two functions come to the rescue – by providing an alphabetical listing of all the module ID's so you can review them and –eventually – plug the ROM directly from the catalogue, for convenience sake.

ROMLIB starts the listing at the top of the list, whereas **CLLIB** prompts for an alphabetical section, A to Z. Choosing "A" here is of course equivalent to executing **ROMLIB**. Both catalogues can run in auto mode or can be stopped using R/S, and then the listing can proceed in manual mode using SST and BST as you can expect.

It is in manual mode where you can use the other shortcuts or "hot keys", as follows:

- **ENTER**^ skips to the next section (or previous if running backwards)
- **[A]** will copy the id# shown to Alpha
- **[P]** will exit the catalog and invoke the **PLUG _** function launcher
- **[SHIFT]** changes the direction of the listing, backwards <-> forwards
- **BackArrow** will cancel out the catalog.

The enumeration terminates in auto mode when the last ROM id# (or first one if running backwards) has been reached. Also keeping any key depressed in RUN mode will halt the sequence displaying until it's released again, so it's easier to keep tabs with the enumeration.

The same considerations made about plugging modules can be made here – be careful not to overwrite anything you're using with a new ROM image, as there's no check whether the target location is already used or not.

As you can imagine there is a lot of code sharing between **ADRID** and these two ROM library catalogue functions. Fundamentally they all use a ROM id# table within the CL-UTILS ROM to look up for the string, and fetch the address in Flash of the corresponding image. This table is quite long, occupying almost 1k in the ROM – yet worth every byte.

The "**A-Z**" prompt entry in **CLLIB** is a refinement of the same idea: it provides a handy shortcut to start your search in the appropriate section, so there's no need to review all the preceding ones – which can be very lengthy considering the sheer number of them, even if you used **ENTER**^ to skip sections. The implementation is quite nice, even if it's the author who says it – have a look at the CLUTILS_Blueprint if you're curious about the MCODE implementation details.

If the section doesn't have any ROM id# starting with such letter (which currently only occurs with [V] and [W] letters) the message "**NO SUCH**" will be shown. Non-alphabetical keys are not valid entries, and will cause the display to just blink and maintain the prompt. Lastly, selecting [X] will list the general-purpose placeholders; refer to the CL manual for details on those.

2.3. HEPAX & SECURE.



2.3.1. Configuring the HEPAX system.

HEPINI	Initializes File System	Author: Howard Owen	
--------	-------------------------	---------------------	--

Use this function to initialize the HEPAX File System on the CL. This is needed on the CL because this feature is disabled in the HEPAX ROM image included in the CL Library, and therefore the addition here.

The function takes two parameters: **the number of HEPAX RAM pages to configure** (in Y) and **the address of the first one** (in X). The procedure consists of writing a few bytes into strategic locations within each HRAM page so that the HEPAX will recognize them as being part of the HEPAX File System. Those locations and byte values are shown in the table below:

Address	Byte value
x000	Page id#
xFE7	Previous HRAM page id# (zero if first)
xFE8	Next HRAM page id# (zero if last)
XFE9	Fixed value = 091
xFED	Fixed value = 090
xFEF	Fixed value = 091
XFF1	Fixed value = 0E5
XFF2	Fixed value = 200

The maximum number of HRAM pages accepted by the function is 9, but typical HEPAX configurations have 2 pages (Standard HEPAX, 8k) or 4 (Advanced HEPAX, 16k). The page id# is assigned starting with "D" for the first page, and increasing it on each contiguous page – up until 15 (hex) in theory.

For this to work the target pages must be mapped to SRAM – or otherwise the byte values could obviously not be changed.

"HPX4"	4k RAM HEPAX Setup	RAM page F, ROM page E	
"HPX8"	8k RAM HEPAX Setup	RAM pages E-F, ROM page D	
"HPX16"	16k RAM HEPAX Setup	RAM pages C-F, ROM page B	

These three functions will prepare the CL ports to hold a properly configured HEPAX file system, starting from the scratch. The process can be divided into four distinct parts:

1. First copying the HEPAX RAM template from Flash into the appropriate number of SRAM blocks, as many times as needed.
2. Followed by mapping those SRAM blocks to the 41 ports, and
3. Then configuring them using **HEPINI** so that they are enabled for the HEPAX ROM to use.
4. Besides that, the functions will also map the HEPX ROM image to the page preceding the first HRAM block, as shown in the table above.

So even if they don't require any input parameter you must be fully aware that the previous MMU mapping to those ports will be overwritten. **The exception being the YFNS ROM itself** – as the programs will check whether it is currently mapped to the page being copied – and abort if that's the case. A nice built-in protection to avoid getting in trouble.

See the appendix 2 for a listing of the FOCAL programs that implement this functionality.

PLUGGX	PLUG Page by X	Page# in X	4k ROMS only
PLUGG _	PLUG page by prompt	Prompts for page: "6-F"	4k ROMS only

Plugging the HEPAX ROM into the appropriate page is accomplished by a single function, using a parameter to define the page address. This function is **PLUGGX**, or "*Plug Page by X*" (and its prompting *doppelgänger* **PLUGG**). Contrary to the port-related convention of the "native" CL functions we're now referring to a page-related one, whereby the arguments of the function are the ROM id# in Alpha (same as usual) and the page# in X – removing the hard-coded dependency of the location used by the **PLUGLxx** and **PLUGUxx** functions.

The picture below (taken from the HEPAX manual) provides the relationship between ports and pages, also showing the physical addresses in the bus and those reserved for special uses (like OS, Timer, Printer, HP-IL, etc). Note that some pages (also called 4k-blocks or simply "blocks") are bank-switched. As always, a picture is worth 1,024 words:

Block Addresses

F	F000-FFFF	Port 4, upper	
E	E000-EFFF	Port 4, lower	
D	D000-DFFF	Port 3, upper	
C	C000-CFFF	Port 3, lower	
B	B000-BFFF	Port 2, upper	
A	A000-AFFF	Port 2, lower	
9	9000-9FFF	Port 1, upper	
8	8000-8FFF	Port 1, lower	
7	7000-7FFF	HP-IL module	
6	6F00-6FFF	Printer	IR printer
5	5000-5FFF	TIME	CX system
4	4000-4FFF	Take-over ROM	
3	3000-3FFF	Unused/CX	
2	2000-2FFF	System ROM 2	
1	1000-1FFF	System ROM 1	
0	0000-0FFF	System ROM 0	

Primary bank Secondary bank

The following error conditions can happen:

- Because of dealing with pages and not full ports, **PLUGGX** will only work with 4k ROMS, or otherwise "*DATA ERROR*" will occur.
- Valid page# inputs are restricted to the 6-F range. Letters other than A-F will be inactive during the prompt, but it will allow any numeric keys - yet values less than 6 will also be rejected, resulting in a "*DATA ERROR*".
- If the string in Alpha is not a valid ROM id# you'll get "*BAD ID*" – as expected.
- If the YFNS ROM is not present (not mapped to the MMU or running on a standard 41 without the CL board) you'll get "*NONEXISTENT*" error.

Note that **PLUGG** and **PLUGG?** are mutually complementary functions, as they both operate on page id# and will take or return the corresponding ROM id# from/to Alpha. You could use **PLUGG?** to interrogate the MMU about page#4, but you can't use **PLUGG** to plug anything to page#4 – there's a dedicated function for that which will be covered in section 2.4 of the manual later on.

2.3.2 Security functions.

The following group of functions are a small detour, in that they aren't directly related to the CL but they come to full fruition when used on this platform.

SECURE	Activate Security	Author: Nick Harmer	Source: Data File
UNLOCK	Deactivate Security	Author: Angel Martin	
XPASS	Change Password	Author: Nick Harmer	Source: Data Fie

Here we have a nice practical application of advanced system control. Use these functions to manage a password-protection scheme for your CL – so nobody without authorized access can use it.

They were published in Data File back in 198x by Nick Harmer, and implemented in Q-RAM devices (a.k.a MLDL). Obvious caveat there was that removing the MLDL from the machine dismantled the whole scheme – but the CL has made it possible as integral part of the core system now.

The protection works as follows:-

1. Function **SECURE** activates the security by setting the protection flag. The execution also switches off the machine. This sets up a process executed on each CALC_ON event, causing to prompt the user for the password during the start-up process.
2. Function **UNLOCK** deactivates the security by clearing the protection flag.
3. Function **XPASS** allows the user to change the password from the default one to his/her favorite one. The length of the password is limited to six (6) characters.

Inputting the password is very simple but very unforgiving as well: at the prompt *"PASSWORD=?"* just type the letters one by one until completing the word, and you're done. If you make a mistake the machine will switch itself off and it'll be "groundhog day" all over gain – until you get it right.

Each keystroke will be acknowledged by a short tone, but no change to the display – so nothing like "*****" as you type the word. If the wrong letter is entered a lower-pitch sound will be heard and the calculator will go to sleep.

Be especially careful when entering a new password code – as there is no repeat input to confirm the entry, so whatever key combination you type will be taken when ending the sequence with R/S. The initial password ("factory default", so to speak) is "CACA".

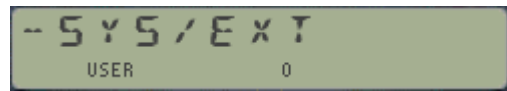


Enter code (up to 6 chrs. long) and end with R/S

Here again it comes without saying that this will only work when the CL-UTILS module is mapped to a SRAM block in the MMU – or otherwise none of the ROM writing will work.

Note: this is how you'd get yourself out of trouble if somehow you forgot the right code: do a memory lost to disable the MMU, then reload the CLUTILS from flash – which has the protection flag cleared. Map it to the right page and enable the MMU again – you're back in charge.

2.4. ADVANCED STUFF



2.4.1. Using Page#4

As mentioned previously page#4 is a special case that requires its own dedicated (un)plugging functions, not covered by PLUGGX or the native (U)PLUG ones either.

PPG#4	Plugs ROM in page#4	Prompts F:L:S	WARNING
UPG#4	Unplugs ROM from p4		

The 41 OS reserves Page #4 as a special location. There are frequent checks done during strategic moments to specific locations that can be used to take control on the system, even over the OS itself if that was required – as it happens with the diagnostics executed from the different SERVICE ROMS.

Because of that, only “take-over” ROMS can be plugged in page#4. They have been written specifically for it and will either take complete control of the system (like the FORTH Module), or drive it from their own directive (like the LAITRAM Module).

Function **PPG#4** prompts for the ROM to plug into the page, options being just those three mentioned above: FORTH, LAITRAM, or SERVICE modules – by their initials: “F:L:S”. Once the selection is made the function transfer execution to a hidden FOCAL program that writes the appropriate entries into the MMU registers, so that the mapping is correct. Refer to the CL manual for details on this.

WARNING: Be aware that once the order is complete you'll be at the mercy of the plugged module. Going back to the “normal” OS may not be as simple as you think, specially with the Service ROM plugged – which requires removing the batteries, then clearing the MMU entry with the MMU disabled after you switch it back on.

For the other instances it is possible to “exit” back to the OS, and thus you could execute **UPPG#4** to unplug the module from the page. Obviously no inputs are needed in this case.

Note that because of their titles being not directly key-able using **XEQ** (an intentional measure) you'll have to use another approach to invoke them. It's a trivial task with the CCD-style CAT'2, either during the catalog run or through a previous assignment to any USER key. Of course as a CL owner you're only one **YPOKE** away from a permanent solution if CLUTILS resides in RAM ☺.

2.4.2. Calculator Flash Backup & Restore.

YFRALL	Backs up to Flash	“OK” or “OKALL” in Alpha	*WARNING*
YFWALL	Restore from Flash	“OK” or OKALL” in Alpha	*WARNING*

The MMU content is preserved during a MEMORY LOST event, and the same is true with the SRAM on the CL board. So using RAM for a complete calculator backup and restore is not a bad idea at all, and it will allow you different setups or complete configurations to be swapped back and forth directly from SRAM.

However SRAM will be erased if the batteries are removed from the calculator for a certain period of time – longer than what it takes to reset a small glitch, but shorter than it used to be for the standard 41, - due to the increased current required to maintain its contents.

Early CL beta user Geoff Quickfall prepared a few FOCAL programs to commit the calculator contents to FLASH, so that *even without the batteries* it'll be preserved for a restore at any later time. It's a powerful concept, but it doesn't come free from pitfalls if you're not careful.

- The first consideration is related to the Flash write function and you should read and understand all about it in the CL manual. Specifically pay strong attention to the recommendations about the battery state before performing any flash-write operation.
- The second one is that **YFWALL** will pick certain hard-coded FLASH locations as destination for the backup, **so the 32k sector 0x0D8000 - 0x0DFFFF will be ERASED** by **YFERASE**. Note that earlier versions of CLUTILS used sector 0x0C8-0x0CF instead. This was moved to the current location to avoid erasing the Solution Books ROMS, added to said sector later on.
- Then there's the question about having to run the programs from RAM for the flash-write/read to work. One could assume that YFNZ is already there but it's much better to make sure that's the case by making a copy on the fly and plugging it to the MMU under program control. Such copy goes to RAM block 0x80C – ***overwriting anything you may have plugged in there previously.***
- Finally the programs also assume that YFNZ is plugged in page#8, that is Lower port 1. Therefore all MMU mapping to YFNS from SRAM and Flash will use that location.

The FOCAL code used by the function is shown below – There is also a check done in MCODE looking for the string "OK" or "OKALL" to be present in Alpha. If none is there the execution will end with **"DATA ERROR"** – as a protection against accidental usage. **"OK"** will get the Calculator content backed up, whilst **"OKALL"** will also include the MMU entries into Flash. Note that on either case the whole 32k sector will be used.

1	LBL "YWALL"	
2	TURBO50	<i>run as fast as possible</i>
3	"062>80C"	<i>Copies YFNS to block</i>
4	YMCPY	<i>in memory 0x80C</i>
5	"80C>RAM"	<i>plugs memory block to port</i>
6	PLUG1L	<i>Must be run from RAM!</i>
7	"0C9000"	<i>Erases 32k sector</i>
8	YFERASE	<i>0x0C8000 - 0x0CFFF</i>
9	"800>0D9"	<i>writes calculator RAM to</i>
10	YFWR	<i>4k block in Flash 0x0D9</i>
11	"804>0DF"	
12	FS? 01	<i>writes MMU entries to</i>
13	YFWR	<i>4k block in Flash 0x0DF</i>
14	"YFNS"	
15	PLUG1L	<i>plugs YFNS from Flash to Port</i>
16	END	<i>we're done.</i>

Should any of those default settings clash with your system setup I'd suggest you change it to match them as the easiest way to go around the incompatibilities. Even if it's possible, *re-writing the program in 41-RAM is strongly not recommended.*

Backing up MMU entries may be seen as superfluous, yet think about the issues arising from restoring MMU configurations that don't include CLUTILS – which is from where the program is being run: welcome to CL-limbo! - Surely something to be avoided.

Note that CLUTILS module may reside in Flash during the process, even if the FOCAL program calls upon **YFWRT** – as the "from-RAM-only" restriction is for YFNS instead.

2.5. DISPLAY UTILS



2.5.1. Alpha and Display Utilities.

The following functions relate to Alpha string manipulation, as the main vehicle for many YFNS functions and are included in the CLUTILS for added convenience. Some

YINPT _	Input Y-String	Prompts for string	
YBSP	Alpha Back Space		Author: W&W GmbH
YCL-	Alpha Delete from "-"		Author: W&W GmbH
YCL>	Alpha Delete from ">"		
YSWAP-	Swap around "-"		
YSWAP>	Swap around ">"		

The reason why characters "-" and ">" are so relevant is the formatting required by many of the YFNZ functions, like **YPEEK**, **YPOKE**, **PLUG_{xx}**, etc. To that effect the most useful function of this group is no doubt **YINPT**, which redefines the keyboard as a hex entry {0-9, A-F}, plus a few special control characters, as follows:

- [J] will add character ">" to the display and Alpha
- [Q] will add character "-" to the display and Alpha
- [M] will add the string "RAM" to the Display and Alpha
- [K] will add the string "16K" to the Display and Alpha
- **BackArrow** will remove the last character (or groups above), or cancel out if Empty
- **ENTER**^ will terminate the entry process and perform **AVIEW**

Using this function expedites the construction of the Alpha strings required by all other Y-Functions, make sure you have it assigned to a handy key as it's likely to be used quite frequently.

DTOA	Display to Alpha		
DTST	Display Test	Author: Chris L. Dennis	Source: PPCJ V18 N8 p14

DTOA is an elusive one to grasp, but basically is the inverse of **AVIEW** – as it copies the characters in the Display to Alpha. The need for this doesn't usually present to the user, as the normal text entry always involves Alpha – but there are times when the reverse is also needed. **DTOA** is used as subroutine by other functions in the module.

As a totally useless demo, assign **DTOA** to any key, then press it in USER mode long enough to see its name shown, then release the key – the words "DTOA" will be copied from the display to Alpha.

DTST Simultaneously lights up all LCD segments and indicators of the calculator display, preceded by all the comma characters (which BTW will be totally unnoticed if your CL is running at 50x Turbo!). Use it to check and diagnose whether your display is fully functional. No input parameters are required.

2.5.2. Other Utilities.

The following functions perform housekeeping tasks and are included in the CLUTILS for added convenience. Some are a remake of the native YFNS with slightly improved behavior, while others just add up for a “rounder pack”.

?MMU	Is the MMU enabled?	No Input	Author: Monte Dalrymple
CDE	HEX string to NNN	String in Alpha	Author: Ken Emery
DCD	NNN to HEX string	NNN in X	Author: W&W GmbH
Y1SEC	1-Second Delay	No input	Author: Monte Dalrymple
YFNZ?	Location for YFNZ	No Input	

Some brief comments follow:

- **?MMU** is almost identical to **MMU?** In the YFNS ROM, but the result in RUN mode is “YES/NO” like the other conditional functions of the machine.
- **Y1SEC** is totally identical to **YSEC**. A candidate for removal in new revisions...
- **YFNZ?** is completely equivalent to **YFNS?**, only that it has different coding. It also must be in the CLUTILS for subroutine purposes. Incidentally, this is how **PLUGGX** checks for YFNS being currently mapped to the target page, and discards the request if so.
- **CDE** and **DCD** are the classic NNN to/from Hex utilities, also used as subroutines throughout the module and thus made available to the user as individual functions as well.

Farewell.

And with this you’ve reached the end of the CLUTILS manual. – I hope these few pages have proven useful to you in your quest to become familiar with its capabilities and whet your appetite for even more to come.

The 41CL is an incredible realization with amazing possibilities, opening the door to yet new developments on the HP-41 platform; all this still happening 33+ years after the original 41 was launched. *Now that’s what I call an achievement!*



© Photo by Geoff Quickfall, 2011

Appendix 1 – Detailed ROM id# table – in alphabetical order.

#	ID	Size	Name	Author / Compiler
1	A41P	12k	Advantage Pac	HP Co.
2	AADV	4k	Advantage Applications	J-F Garnier
3	ADV1	16k	Adventure_1	Angel Martin
4	ADV2	12k	Adventure_2	Angel Martin
5	AEC3	8K	AECROM 13-digit	Angel Martin
6	AECR	8k	AECROM	Red Shift
7	AFDE	8k	AFDC1	GunZen
8	AFDF	8k	AFDC2	GunZen
9	AFIN	4k	Auto Finance	GMAC
10	ALGG	8k	Algebra ROM	Angel Martin
11	ALGY	4k	Astro*ROM	Elgin Knowles & Senne
12	ALPH	4k	ALPHA ROM	A. Martin & D. Wilder
13	AOSX	4k	AMC OS/X	Angel Martin
14	ASM4	4k	Assembler4	??
15	ASMB	4k	Assembler3	??
16	ASTT	16k	ASTRO-2010 Module	Jean-Marc Baillard
17	AUTO	4k	Auto-Start / Dupl ROM	HP Co.
18	AV1Q	4k	AV1 ROM	Beechcraft
19	AVIA	4k	Aviation Pac	HP Co.
20	B52B	8k	B-52 ROM	Boeing
21	BCMW	4k	BCMW ROM	??
22	BESL	8k	Bessel ROM	A. Martin & JM Baillard
23	BLDR	8k	BLD ROM	W. Doug Wilder
24	BLND	4k	Bufferland ROM	Angel Martin
25	CCDP	8k	CCD Plus	Angel Martin
26	CCDR	8k	CCD Module	W&W GmbH
27	CCDX	4k	CCD OS/X	Raymond del Tondo
28	CHEM	4k	Chemistry User ROM	??
29	CHES	8k	Chess/Rubik's ROM	Claude Roetlgen
30	CIRC	4K	Circuit Analysis Pac	HP Co.
31	CLIN	4K	Clinical Lab Pac	HP Co.
32	CLUT	4k	CL Utilities	Angel Martin
33	CURV	8k	Curve-Fitting Module	Angel Martin
34	CVPK	8k	Cv-Pack ROM	??
35	DA4C	4k	DisAssembler 4C	W. Doug Wilder
36	DACQ	8k	Data Acquisition Pac	HP Co.
37	DASM	4k	DisAssembler 4D	W. Doug Wilder
38	DAVA	4K	David Assembler 2C	David van Leeuwen
39	DEVI	8k	HP-IL Development	HP Co.
40	DIIL	4k	HP-IL Diagnostics	HP Co.
41	DMND	4k	Diamond ROM	??
42	DYRK	4k	Dyerka ROM	David Yerka
43	E41S	8k	ES41 Module	Eramco
44	ESML	4k	ES MLDL 7B	Eramco
45	EXIO	4k	Extended I/O Module	HP Co.
46	EXTI	4k	Extended-IL ROM	Ken Emery
47	FACC	4k	300889_FACC	??
48	FINA	4k	Financial Pac	HP Co.
49	FRTH (*)	8k	FORTH Module	Serge Vaudenay
50	FUNS	8k	Fun Stuff Module	Angel Martin
51	GAME	4k	Games Pac	HP Co.
52	GMAS	4k	Auto Fiance-2 Module	GMAC
53	GMAT	8k	Auto Fiance-3 Module	GMAC
54	HCMP	4k	HydraComp ROM	Paul Monroe
55	HEPR	4k	HEPAX RAM Template	VM Electronics

56	HEPX	16k	HEPAX Module	VM Electronics
57	HOME	4k	Home Management. Pac	HP Co.
58	ICDO	4k	Icode ROM	??
59	IDC1	8k	ML-ICD	BCMC 1987
60	IDC2	4k	BG/UG IDC	BCMC 1985
61	JMAT	8k	JMB Math	Jean-Marc Baillard
62	JMTX	8k	JMB Matrix	Jean-Marc Baillard
63	ILBF	4k	IL-Buffer	Angel Martn
64	KC135	12k	Weight & Balance Comp.	??
65	L119	8k	AFDC-1E-003	Zengun
66	LAIT (*)	4k	LaitRAM XQ2	LaitRam Corp.
67	LAND	4k	Land Navigation ROM	Warren Furlow
68	LBLS	4k	Labels ROM	W. Doug Wilder
69	MADV	12k	Modified Advantage ROM	Angel Martn
70	MATH	4k	Math Pac	HP Co.
71	MCHN	4k	Machine Construction Pac	HP Co.
72	MDP1	8k	AFDC-1F ROM	Zengun
73	MDP2	8k	AFDC-1F ROM	Zengun
74	MELB	4k	Melbourne ROM	PPC Members
75	MILE	8k	Military Engineering ROM	??
76	MLBL	4k	Mainframe Labels	David van Leeuwen
77	MLRM	4K	ML ROM	Frits Ferwerda
78	MLTI	8k?	Multi-Prec. Library	Peter Platzter
79	MTRX	4k	MATRIX ROM	Angel Martin
80	MTST	4k	MC Test ROM	??
81	MUEC	8k	Muecke ROM	Mücke Software GmbH
82	NAVI	8k	Navigation Pac	HP Co.
83	NCHP	4k	NoVoCHAP	G. Isene & A. Martin
84	NFCR	4k	NFC ROM	Nelson F. Crowe
85	NPAC	8k	NavPac ROM	??
86	NVCM	8k	NaVCOM 2	??
87	OILW	8k	OilWell Module	Jim Daly
88	P3BC	16k	Aviation for P3B/C	??
89	PANA	8k	PANAME ROM	S. Barizienne & JJ Dhenin
90	PARI	4k	PARIO ROM	Nelson F. Crowe
91	PCOD	4k	Proto-Coder 1A	Nelson F. Crowe
92	PETR	8k	Petroleum Pac	HP Co.
93	PLOT	8k	Plotter Module	HP Co.
94	PMLB	4k	PPC Melb ROM	PPC Members
95	POLY	8k	Polynomial Analysis	A. Martin & JM Baillard
96	PPCM	8k	PPC ROM	PPC Members
97	PRFS	4k	ProfiSet	Winfried Maschke
98	PRIQ	8k	PRIDE ROM	??
99	QUAT	8k	Quaternion ROM	Jean-Marc Baillard
100	RAMP	4k	RAMPage Module	Angel Martin
101	REAL	8k	Real State Pac	HP Co.
102	ROAM	4k	ROAM Module	Wilson B. Holes
103	ROMS	4k	SV's ROM	Serge Vaudenay
104	SANA	12k	SandMath-12k	Angel Martin
105	SBOX	8k	SandBox	Angel Martin
106	SEAK	4k	SeaKing MK5	Navy Air
107	SECY	4k	Securities Pac	HP Co.
108	SGSG	4k	Gas Module	SGS Redwood
109	SIMM	16k	SIM Module	??
110	SKWD	4k	Skwid's BarCode	Ken Emery
111	SMCH	8k	Speed Machine	Alameda Mngmt. Corp.
112	SMPL	4k	Simplex Module	Phillipe J. Roussel
113	SMTS	8k	SandMath-8k	Angel Martin

114	SND2	8k	SandMath-II	Angel Martin
115	SPEC	4k	Spectral Analysis	Jean-Marc Baillard
116	SRVC (*)	4k	Service ROM	HP Co.
117	STAN	4k	Standard Pac	HP Co.
118	STAT	4k	Statistics Pac	HP Co.
119	STRE	4k	Stress Analysis Pac	HP Co.
120	STRU	8k	Structural An, Pac	HP Co.
121	SUPR	8k	SUP-R-ROM	James W. Vick
122	SURV	4k	Surveying Pac	HP Co.
123	THER	4k	Thermal Pac	HP Co.
124	TOMS	4k	Tom's ROM	Thomas A. Bruns
125	TOOL	4k	ToolBox-II	Angel Martin
126	TREK	4k	Start Trek	Angel Martin
127	TRIH	4k	83Trinh	Phil Trinh
128	UNIT	4k	Unit Conversion	Angel Martin
129	USPS	8k	Mail Delivery	USPS
130	XXXA	4k	Empty	Not listed
131	XXXB	4k	Empty	Not listed
132	XXXC	4k	Empty	Not listed
133	XXXD	8k	Empty	Not listed
134	XXXE	8k	Empty	Not listed
135	XXXF	16k	Empty	Not listed
136	YFNS	4k	Alternate YFNS	Monte Dalrymple
137	YFNZ	4k	Main YFNS	Monte Dalrymple
138	Z41Z	8k	41Z Module	Angel Martin
139	ZENR	4k	Zenrom	Zengrange Ltd.
140	ZEPR	4k	Programmer	Zengrange Ltd.

(*) Take-over ROMS

Other modules not included in the Library:-

For sure many more of these abound, yet these are the ones I have knowledge of – feel free to complete the list with your own entries, and don't forget to share it with the whole community.

- | | | |
|--------------------------------|-------|-------------------------|
| 1. CCD Advanced Apps. | 4k | Ángel Martin |
| 2. Geometry 2011 | 4k | Jean-Marc Baillard |
| 3. Market Forecast | 4k | Forecaster? |
| 4. MONOPOLY ROM | 8k | Thomas Rodke |
| 5. Mortar Fire Data Calculator | 8k | MDN Canada |
| 6. Mountain Computer EPROM | 4k | Paul Lind |
| 7. Dr. Z RaceTrack Module | 4k | William T. Ziemba |
| 8. SNEAP1/2/3 | 3x 8k | SNEAP Society (F) |
| 9. SUDOKU & Sound | 4k | JM Baillard & Á. Martin |
| 10. VECTOR Analysis | 4k | Ángel Martin |
| 11. Yach Computer | 4k | Bobby Schenk |

Modules included in Flash without Module ID#

- | | | | | | |
|------------------------|-------|-------------|----------------------|-------|-------------|
| 1. ISENE ROM | 0x0C9 | Geir Isene | 8. Antennas | 0x0D1 | HP Co. |
| 2. Bus Sales/Mkt/Stat. | 0x0CA | HP Co. | 9. Optometry I & II | 0x0D2 | HP Co. |
| 3. Control Systems | 0x0CB | HP Co. | 10. Physics | 0x0D3 | HP Co. |
| 4. Electrical Eng. | 0x0CC | HP Co. + ÁM | 11. Geometry | 0x0D4 | HP Co. + ÁM |
| 5. Lend, Lease & Sav. | 0x0CD | HP Co. | 12. High-Level Math | 0x0D5 | HP Co. |
| 6. Test Statistics | 0x0CE | HP Co. | 13. Interchang. Sol. | 0x0D6 | UPLE |
| 7. Mechanical Eng. | 0x0CF | HP Co. + ÁM | 14. Module Database | 0x0D7 | MD |

Appendix 2. FOCAL program Listings.

Provided for your reference and in case you feel like experimenting with your own settings.

As always, mind the potential conflicts with other modules when plugging stuff, and pay special attention not to overwrite YFNS. (you're safe if using **PLUGGX** – it won't let you to :-)

In the HEPAX configuration code the role of **HEPINI** is to write the appropriate words into the HRAM pages, as per the description provided before. This could also be done using **YPOKE**, but the memory requirements are much larger due to all the alpha strings that would be required to do so.

For example, see below for the 16k case, using pages C,D,E, and F.

This would mean having to write on each page the four page id#s, plus the pointers to the previous and next pages, for a total of 10x – or equivalent to 110 bytes:

"809FE7-000C"

"808000-000C"

"808FE8-000D"

"80AFE7-000D"

"809000-000D"

"809FE8-000E"

"80BFE7-000E"

"80A000-000E"

"80AFE8-000F"

"80B000-000F"

01	LBL "HPX4"	Entry for 4k
02	E	
03	GTO 02	
04	LBL "HPX8"	Entry for 8k
05	2	
06	GTO 02	
07	LBL "HPX16"	Entry for 16k
08	0	
09	LBL 02	
10	X<>F	flag setting
11	TURBO50	run as fas as possible
12	"0B9>808"	prepare for 0x808
13	AVIEW	provide feedback
14	YMCPY	copy to 0x808 in RAM
15	FS? 00	4k case?
16	GTO 00	yes, skip the rest
17	YBSP	remove last chr
18	"9"	replace it
19	AVIEW	provide feedback
20	YMCPY	copy to 0x809 in RAM
21	FS? 01	8k case?
22	GTO 00	yes, skip the rest
23	YBSP	remove last char
24	"A"	replace it
25	AVIEW	provide feedback
26	YMCPY	copy to 0x80A in RAM
27	YBSP	remove last char
28	"B"	replace it
29	AVIEW	provide feedback
30	YMCPY	copy to 0x80B in RAM
31	LBL 00	
32	"SETUP..."	announce last phase
33	AVIEW	
34	"-RAM"	buffer common string
35	ASTO L	in temporary storage
36	"808"	build first mapping text
37	ARCL L	
38	PLUG4U	plug 0x808 to page#F
39	FS? 00	4k case?
40	GTO 01	yes, jump over
41	"809"	build 2nd. Mapping text
42	ARCL L	
43	PLUG4L	plug 0x809 to page#E
44	FS? 01	8k case?
45	GTO 03	yes, jump over
46	"80A"	build 3er. Mapping text
47	ARCL L	
48	PLUG3U	plug 0x80A to page#D
49	"80B"	build 4th. Mapping text
50	ARCL L	
51	PLUG3L	plug 0x80B to port#C
52	4	configuration parameters:
53	ENTER^	four pages
54	12	starting at page#C
55	GTO 00	
56	LBL 03	8k case
57	2	config parm:
58	ENTER^	two pages
59	14	starting at page#E
60	GTO 00	
61	LBL 01	4k case
62	E	config parm:
63	ENTER^	just one page,
64	15	starting at page#F
65	LBL 00	
66	HEPINI	configure HEPAX FileSys
67	E	get page# below
68	-	
69	"HEPX"	
70	PLUGGX	plug HEPAX there
71	HEPDIR	show we've done it
72	END	done.

Appendix 3.- MCODE Listing showing the Alphabetical sections prompting code.

The function CLLIB begins by building the prompt text in the display. Using the OS routine [PROMF2] is helpful to save bytes, so there's no need to write the function name again, "CLLIB". Alpha is cleared using [CLA], just to prepare for a possible copy of the ROM id# to Alpha using the [A] hot-key in run mode. Then we get into a partial data entry "condition", waiting for a key to be pressed.

Back Arrow sends the execution to [EXIT3], to do the housekeeping required to reset everything back to the standard OS-required status (disable Display, resetting Keyboard bits, CPU flags, etc.).

Since the valid keys are quite a lot [A-Z] we need to use multiple conditions in the logic. The first two rows are the easiest; as they set up CPU flag#4 and that can be tested easily. In this case we copy the mantissa sign in A to C[S&X], then store it in B[S&X] and we move on.

1	CLLIB	BCKARW	A66E	341	PORT DEP:		
2	CLLIB		A66F	08C	GO		
3	CLLIB		A670	36E	->A36E		[EXIT3]
4	CLLIB	Header	A671	082	"B"		
5	CLLIB	Header	A672	009	"I"		
6	CLLIB	Header	A673	00C	"L"		
7	CLLIB	Header	A674	00C	"L"		CL Library
8	CLLIB	Header	A675	003	"C"		Prompting - Alphabetical
9	CLLIB	CLLIB	A676	000	NOP		
10	CLLIB		A677	158	M=C ALL		
11	CLLIB		A678	345	?NC XQ		Clears Alpha
12	CLLIB		A679	040	->10D1		[CLA]
13	CLLIB		A67A	3C1	?NC XQ		Enable & Clear Display
14	CLLIB		A67B	0B0	->2CF0		[CLLCDE]
15	CLLIB		A67C	198	C=M ALL		fst id#
16	CLLIB		A67D	34D	?NC XQ		
17	CLLIB		A67E	014	->05D3		[PROMF2]
18	CLLIB		A67F	3BD	?NC XQ		
19	CLLIB		A680	01C	->07EF		[MESSL]
20	CLLIB		A681	001	"A"		
21	CLLIB		A682	02D	"."		
22	CLLIB		A683	21A	"Z"		
23	CLLIB		A684	115	?NC XQ		Partial Data Entry!
24	CLLIB		A685	038	->0E45		[NEXT1]
25	CLLIB		A686	343	JNC -24d		
26	CLLIB		A687	04C	?FSET 4		rows 1 & 2?
27	CLLIB		A688	063	JNC +12d		
28	CLLIB		A689	130	LDI S&X		
29	CLLIB		A68A	00A	CON:		
30	CLLIB		A68B	35E	?A#0 MS		was "J" pressed?
31	CLLIB		A68C	023	JNC +04		yes, skip next
32	CLLIB		A68D	046	C=0 S&X		
33	CLLIB		A68E	0BE	A<=>C MS		
34	CLLIB		A68F	2FC	RCR 13		
35	CLLIB		A690	0E6	C<=>B S&X		save chr# in B[S&X]
36	CLLIB		A691	369	PORT DEP:		Transfer to next section
37	CLLIB		A692	03C	GO		
38	CLLIB		A693	2E9	->A6E9		[MOVEON]
39	CLLIB		A694	0B0	C=N ALL		pressed keycode
40	CLLIB		A695	106	A=C S&X		save in A[S&X] for comparisons
41	CLLIB		A696	21C	PT= 2		adjust pointer
42	CLLIB		A697	05A	C=0 M		reset counter
43	CLLIB		A698	130	LDI S&X		upper value
44	CLLIB		A699	01A	"Z"		from there down!
45	CLLIB		A69A	1BC	RCR 11		put it in C[M]
46	CLLIB		A69B	013	JNC +02		skip over line
47	CLLIB		A69C	343	JNC -24d		pipe-lining up

For the rest [K-Z] we'll need to read the keycode of the pressed key and act accordingly. Also *we need to discard any non-letter key*, rejecting it if its keycode value is outside of the [A,Z] range.

Now the show is about to start: see how the key pressed value (in N) is compared with every possible value in the [K-Z] range, building the "pointer" in C[S&X] by repeat one-additions until coming up to its final result.

48	CLLIB		A69D	130	LDI S&X	
49	CLLIB		A69E	120	CON:	XEQ keycode [120]
50	CLLIB		A69F	366	?A#C S&X	
51	CLLIB		A6A0	1C3	JNC +56d	[K]
52	CLLIB		A6A1	222	C=C+1 @PT	keycode [220]
53	CLLIB		A6A2	366	?A#C S&X	
54	CLLIB		A6A3	1B3	JNC +54d	[L]
55	CLLIB		A6A4	222	C=C+1 @PT	keycode [320]
56	CLLIB		A6A5	366	?A#C S&X	
57	CLLIB		A6A6	1A3	JNC +52d	[M]
58	CLLIB		A6A7	130	LDI S&X	
59	CLLIB		A6A8	030	CON:	ENTER^ keycode [030]
60	CLLIB		A6A9	366	?A#C S&X	
61	CLLIB		A6AA	18B	JNC +49d	[N]
62	CLLIB		A6AB	222	C=C+1 @PT	keycode [130]
63	CLLIB		A6AC	222	C=C+1 @PT	keycode [230]
64	CLLIB		A6AD	366	?A#C S&X	
65	CLLIB		A6AE	173	JNC +46d	[O]
66	CLLIB		A6AF	222	C=C+1 @PT	keycode [330]
67	CLLIB		A6B0	366	?A#C S&X	
68	CLLIB		A6B1	163	JNC +44d	[P]
69	CLLIB		A6B2	130	LDI S&X	
70	CLLIB		A6B3	040	CON:	"-" keycode [040]
71	CLLIB		A6B4	366	?A#C S&X	
72	CLLIB		A6B5	14B	JNC +41d	[Q]
73	CLLIB		A6B6	222	C=C+1 @PT	keycode [140]
74	CLLIB		A6B7	366	?A#C S&X	
75	CLLIB		A6B8	13B	JNC +39d	[R]
76	CLLIB		A6B9	222	C=C+1 @PT	keycode [240]
77	CLLIB		A6BA	366	?A#C S&X	
78	CLLIB		A6BB	12B	JNC +37d	[S]
79	CLLIB		A6BC	222	C=C+1 @PT	keycode [340]
80	CLLIB		A6BD	366	?A#C S&X	
81	CLLIB		A6BE	11B	JNC +35d	[T]
82	CLLIB		A6BF	130	LDI S&X	
83	CLLIB		A6C0	050	CON:	"+" keycode [050]
84	CLLIB		A6C1	366	?A#C S&X	
85	CLLIB		A6C2	103	JNC +32d	[U]
86	CLLIB		A6C3	222	C=C+1 @PT	keycode [150]
87	CLLIB		A6C4	366	?A#C S&X	
88	CLLIB		A6C5	0F3	JNC +30d	[V]
89	CLLIB		A6C6	222	C=C+1 @PT	keycode [250]
90	CLLIB		A6C7	366	?A#C S&X	
91	CLLIB		A6C8	0E3	JNC +28d	[W]
92	CLLIB		A6C9	222	C=C+1 @PT	keycode [350]
93	CLLIB		A6CA	366	?A#C S&X	
94	CLLIB		A6CB	0D3	JNC +26d	[X]
95	CLLIB		A6CC	130	LDI S&X	
96	CLLIB		A6CD	060	CON:	"*" keycode [060]
97	CLLIB		A6CE	366	?A#C S&X	
98	CLLIB		A6CF	0BB	JNC +23d	[Y]
99	CLLIB		A6D0	222	C=C+1 @PT	keycode [160]
100	CLLIB		A6D1	366	?A#C S&X	
101	CLLIB		A6D2	0AB	JNC +21	[Z]
102	CLLIB		A6D3	265	?NC XQ	Blink Display - pass #1
103	CLLIB		A6D4	020	->0899	[BLINK1]
104	CLLIB		A6D5	265	?NC XQ	Blink Display - pass #2
105	CLLIB		A6D6	020	->0899	[BLINK1]
106	CLLIB		A6D7	22B	JNC -59d	ONE PROMPT
107	CLLIB	[K]	A6D8	27A	C=C-1 M	
108	CLLIB	[L]	A6D9	27A	C=C-1 M	
109	CLLIB	[M]	A6DA	27A	C=C-1 M	
110	CLLIB	[N]	A6DB	27A	C=C-1 M	
111	CLLIB	[O]	A6DC	27A	C=C-1 M	
112	CLLIB	[P]	A6DD	27A	C=C-1 M	
113	CLLIB	[Q]	A6DE	27A	C=C-1 M	
114	CLLIB	[R]	A6DF	27A	C=C-1 M	
115	CLLIB	[S]	A6E0	27A	C=C-1 M	
116	CLLIB	[T]	A6E1	27A	C=C-1 M	
117	CLLIB	[U]	A6E2	27A	C=C-1 M	
118	CLLIB	[V]	A6E3	27A	C=C-1 M	
119	CLLIB	[W]	A6E4	27A	C=C-1 M	
120	CLLIB	[X]	A6E5	27A	C=C-1 M	
121	CLLIB	[Y]	A6E6	27A	C=C-1 M	
122	CLLIB	[Z]	A6E7	03C	RCR 3	place it in C[S&X]
123	CLLIB		A6E8	0E6	C<>B S&X	save chr# in B[S&X]

The last part is about presenting the chosen key – allowing NULLing if it's held down long enough – Resetting everything back to normal conditions [CLNUP], and see whether there actually exists such a section – before we launch into a blindfold enumeration. This is done by the subroutine [SRCHR], which will fetch the address in the ROM id #table where the section starts. With that we'll transfer the execution to the **ROMLIB** function code where the actual enumeration will take place - only with a padded value to start from, as opposed to doing it from the top of the table.

124	CLLIB	MOVEON	A6E9	379	PORT DEP:	CleanUp and Show
125	CLLIB		A6EA	03C	XQ	Allows NULLing
126	CLLIB		A6EB	261	->A661	[CLNUP]
127	CLLIB		A6EC	149	?NC XQ	Disable PER, enable RAM
128	CLLIB		A6ED	024	->0952	[ENCP00]
129	CLLIB		A6EE	215	?NC XQ	Reset BIT sequence
130	CLLIB		A6EF	00C	->0385	[RSTSQ]
131	CLLIB		A6F0	130	LDI S&X	Location of first ID
132	CLLIB		A6F1	32B	<start of search>	[romtbl]
133	CLLIB		A6F2	106	A=C S&X	save offset in A[S&X]
134	CLLIB		A6F3	349	PORT DEP:	Search Char in B[S&X]
135	CLLIB		A6F4	08C	XQ	start offset in A[S&X]
136	CLLIB		A6F5	353	->A353	[SRCHR]
145	CLLIB		A6F6	023	JNC +04	not found, bail out with style
146	CLLIB		A6F7	341	PORT DEP:	Transfer code to Main LIB
147	CLLIB		A6F8	08C	GO	
148	CLLIB		A6F9	386	->A386	[MERGE]
149	CLLIB	INFMSG	A6FA	261	?NC XQ	debounce keyboard
150	CLLIB		A6FB	000	->0098	[RSTKB]
151	CLLIB		A6FC	3B5	PORT DEP	Display Message
152	CLLIB		A6FD	08C	XQ	Unless Error Flag is set.
153	CLLIB		A6FE	3EE	->AFEE	[DSPERR]
154	CLLIB		A6FF	00E	"N"	
155	CLLIB		A700	00F	"O"	
156	CLLIB		A701	020	" "	
157	CLLIB		A702	013	"S"	
158	CLLIB		A703	015	"U"	
159	CLLIB		A704	003	"C"	
160	CLLIB		A705	208	"H"	
161	CLLIB		A706	389	PORT DEP	Output Message
162	CLLIB		A707	08C	GO	
163	CLLIB		A708	016	->A816	[APEREX]

Note how [SRCHR] is really part of the **ADRID** function code, which also does table look-ups for its own purpose. This code is written around the table structure; refer to the Blueprints for more details.

32	ADRID	SRCHR	A353	04E	C=0 ALL	Expects chr# in B[S&X]
33	ADRID		A354	35D	?NC XQ	
34	ADRID		A355	000	->00D7	[PCTOC]
35	ADRID		A356	03C	RCR 3	get page number
36	ADRID		A357	130	LDI S&X	
37	ADRID		A358	300	CON:	
38	ADRID		A359	1E6	C=C+C S&X	double it up
39	ADRID		A35A	206	C=C+A S&X	add offset to it
40	ADRID		A35B	1BC	RCR 11	put it in C(6:3)
41	ADRID		A35C	0BA	A<>C M	put it in A(6:3)
42	ADRID		A35D	066	A<>B S&X	put reference in A[S&X]
43	ADRID	NXTADR	A35E	0BA	A<>C M	bring adr to C[M]
44	ADRID		A35F	11A	A=C M	keep it in A[M]
45	ADRID		A360	330	FETCH S&X	read value at address
46	ADRID		A361	2E6	?C#0 S&X	value non-zero?
47	ADRID		A362	3A0	?NC RTN	NO, TERMINATE HERE!
48	ADRID		A363	366	?A#C S&X	are they different?
49	ADRID		A364	033	JNC +06	no, -> [FOUND]
50	ADRID		A365	05A	C=0 M	add offset: 5 BYTES
51	ADRID		A366	01C	PT=3	
52	ADRID		A367	150	LD@PT- 5	
53	ADRID		A368	15A	A=A+C M	next addr field
54	ADRID		A369	3AB	JNC -11d	loop back
55	ADRID	FOUND	A36A	1B0	POPADR	
56	ADRID		A36B	23A	C=C+1 M	
57	ADRID		A36C	170	PUSHADR	increase RTN adr
58	ADRID		A36D	3E0	RTN	

And that's all folks - easy when you know the tricks ☺

Appendix 4.- Serial Transfer CLWRITE source code. – written by Raymond Wiker.

```
using System;
using System.IO;
using System.IO.Ports;
using System.Threading;

public class CLWriter
{
    public static void Main(string [] args)
    {
        int baudrate = 1200;
        int delay = 0;
        if (args.Length < 2) {
            Console.Error.WriteLine("Usage:");
            Console.Error.WriteLine("  {0} file port [baudrate [delay]]", "CLWriter");
            Console.Error.WriteLine();
            Console.Error.WriteLine("Where baud defaults to {0}", baudrate);
            Console.Error.WriteLine("and delay defaults to {0}", delay);
            Console.Error.WriteLine("Available Ports:");
            Console.Error.WriteLine();
            foreach (string s in SerialPort.GetPortNames())
            {
                Console.Error.WriteLine("  {0}", s);
            }
            return;
        }
        string filename = args[0];
        string portname = args[1];
        if (args.Length > 2) {
            baudrate = int.Parse(args[2]);
            if (baudrate != 1200 && baudrate != 2400 &&
                baudrate != 4800 && baudrate != 9600) {
                Console.Error.WriteLine("Invalid baudrate {0}; should be one of", baudrate);
                Console.Error.WriteLine("1200, 2400, 4800, 9600");
                return;
            }
        }
        if (args.Length > 3) {
            delay = int.Parse(args[3]);
            if (delay > 10) {
                Console.Error.WriteLine("delay {0} probably too large.", delay);
                return;
            }
        }

        if (!File.Exists(filename)) {
            Console.Error.WriteLine("File {0} does not exist.", filename);
            return;
        }

        FileStream fstream = File.Open(filename, FileMode.Open);

        if (fstream.Length > 8192) {
            Console.Error.WriteLine("WARNING: {0} is over 8192 bytes long ({1});", filename,
fstream.Length);
            Console.Error.WriteLine("Will only transfer the first 8192 bytes.");
        }
    }
}
```

```

    }

    BinaryReader binReader = new BinaryReader(fstream);

    SerialPort serialport = new SerialPort();
    serialport.PortName = portname;
    serialport.BaudRate = baudrate;
    serialport.Parity = Parity.None;
    serialport.DataBits = 8;
    serialport.StopBits = StopBits.One;
    serialport.Handshake = Handshake.None;

    serialport.Open();

    try {
        byte[] buffer = new byte[8192];
        int count = binReader.Read(buffer, 0, 8192);

        // swap high & low bytes:
        for (int i = 0; i < count; i += 2) {
            byte tmp = buffer[i];
            buffer[i] = buffer[i+1];
            buffer[i+1] = tmp;
        }

        for (int i = 0; i < count; i++) {
            Console.Write("{0:x2} ", buffer[i]);
            if (i % 16 == 15) {
                Console.WriteLine();
            }
            serialport.Write(buffer, i, 1);
            if (delay > 0) {
                Thread.Sleep(delay);
            }
        }
        Console.WriteLine();
    }
    catch (EndOfStreamException) {
        // nada
    }
    serialport.Close();
}
}

```


1) Copy YFNS-1A to RAM at 80C000, and patch for items 2, 5, 8. These affect the operation of YIMP. I did this with a variation of the PATCHIT program posted earlier.

2) Execute TURBO50.

3) Execute SERINI

4) Execute BAUD12. From the documentation, this should not be necessary, but I had to explicitly set 1200 baud to get the transfer to work.

5) The file yfns-1e.rom has the opposite byte order of what YIMP expects, so the transfer program needs to perform byte swapping. Alternatively, you might do the byte-swapping before you do the transfer.

6) Transfer the ROM; I chose to transfer it to 80D000 (i.e, put 80D000-0FFF in the alpha register, start YIMP). For the transfer, I used the CLWriter program that I posted a few days back, with the command

```
CLWriter.exe yfns-1e-fixed.rom com1 1200 5
```

--- the file yfns-1e-fixed.rom is the byte-swapped version of yfns-1e.rom. I probably should have chosen a slightly different name, but that does not really matter. The "5" means that I put a 5 millisecond delay after each byte. It may not actually be necessary; I added it because I got timeouts, but these were probably because I left out step 4 (BAUD12).

7) Execute PLUG1L with "80D-RAM" in the Alpha register.

8) Verify (using CATALOG 2) that I'm now running YFNS-1E.