CommandRef.hyper

COLLABORATORS

	TITLE : CommandRef.hyper	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 26, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

CommandRef.hyper

1.1 Command Reference (Wed Jul 15 08:40:38 1992)

```
                Contents:

                 Introduction

                 Commands sorted by function
                Commands:

                 account
                  (task usage, stack check)

                 active
                  (make logical window active)

                 addfunc
                  (add function to monitor)

                 addstruct
                  (add structure definitions)

                 addtag
                  (give type to memory range)

                 alias
                    (set an alias)

                 assign
                  (ARexx: assign to variable)

                 async
                    (ARexx: undo Sync command)

                 attach
                  (attach macro to key)
  attc      (go to the macro list)

                 awin
                    (open/close the Rexx logical window)
```

```
                break
                   (control breakpoints)

                checktag
                 (check type of address)

                cleanup
                 (cleanup allocated memory)

                clear
                   (clear memory range with value)

                cleartags
                 (clear all types for memory ranges)

                clip
                   (ARexx: set or get clip)

                closedev
                 (close PowerVisor device)

                closelw
                 (close logical window)

                closepw
                 (close physical window)

                closescreen
                 (close screen and windows)

                closewindow
                 (close window)

                cls
                   (clear current logical window)

                color
                   (set RGB values for PowerVisor screen)

                colrow
                 (set size of logical window)
conf       (go to autoconfig list)

                copy
                   (copy memory)

                crash
                   (patch all tasks in system)
crsh       (go to list with crashed tasks)

                curdir
                 (set a new current directory for process)

                current
                 (set the current logical window)
dbug       (go to the debug list)
```

```
                debug
                   (control debugging)

                devcmd
                 (give device command)

                devinfo
                 (give information about PowerVisor device)
devs      (go to Exec device list)

                disp
                   (display integer)
dosd      (go to Dos device list)

                drefresh
                 (refresh debug display)

                dscroll
                 (scroll debug logical window)

                dstart
                 (set position in debug logical window)

                duse
                   (set current debug task)

                dwin
                   (open/close the debug logical window)

                error
                   (generate error string)

                event
                   (generate an input event)
exec      (go to ExecBase list)
fdfi      (go to fd-file list)

                fill
                   (fill memory with string)
fils      (go to the file list)

                fit
                   (fit a logical window to the visible size)

                float
                   (change a floating point register)
font      (go to the font list)

                for
                   (execute command for each element in list)

                freeze
                 (freeze a task or process)

                fregs
                   (show floating point registers)
```

```
                front
                   (bring PowerVisor screen to front)
func      (go to the monitor function list)

                gadgets
                 (list all gadgets for a window)

                getstring
                 (get a string from the user)

                go
                   (start executing machinelanguage)
graf      (go to the GfxBase list)

                help
                   (give online help)

                hide
                   (ARexx: hide output from ARexx)

                hold
                   (close PowerVisor screen and windows)

                home
                   (go to home position in current logical window)

                hunks
                   (show all hunks for a process)
ihan      (go to the input handler list)

                info
                   (give information about list element)

                interprete
                 (interprete some data as a structure)
intb      (go to the IntuitionBase list)
intr      (go to the Exec interrupt list)

                kill
                   (kill a task or process)

                led
                   (toggle powerled)

                libfunc
                 (give information about library function)

                libinfo
                 (give information about library function)
libs      (go to the library list)

                list
                   (list the current list)

                llist
                   (traverse list)
```

```
                load
                   (load a file to memory)

                loadfd
                 (load a fd-file)

                loadtags
                 (load definitions for memory)

                locate
                 (set cursor position on logical window)
lock     (go to the lock list)

                log
                   (log output to file)
lwin     (go to the logical window list)

                memory
                 (dump memory)
memr     (go to the memory list)

                memtask
                 (give memory of task)

                mmuregs
                 (show MMU registers)

                mmureset
                 (reset MMU tree)

                mmurtest
                 (test address for read)

                mmutree
                 (show MMU tree)

                mmuwtest
                 (test address for write)

                mode
                   (change PowerVisor preferences)
moni     (go to monitor list)

                move
                   (move a physical window)

                next
                   (continue searching)

                on
                   (execute command on logical window)

                opendev
                 (open a PowerVisor device)

                openlw
                 (open a logical window)
```

```
                openpw
                 (open a physical window)

                owin
                   (open/close PortPrint logical window)

                owner
                   (show owner of memory)

                pathname
                 (show pathname from lock)
port      (go to message port list)

                prefs
                   (change PowerVisor preferences)

                print
                   (print string)
pubs      (go to public screen list)

                pvcall
                 (access internal structures)
pwin      (go to physical window list)

                quit
                   (quit PowerVisor or script)

                rblock
                 (read block from trackdisk.device)

                refresh
                 (control refresh rate and command)

                regs
                   (show registers for task)

                rematttach
                 (remove macro from key)

                remclip
                 (ARexx: remove clip)

                remcrash
                 (remove crashed task)

                remfunc
                 (remove monitor function)

                remhand
                 (remove input handler)

                remove
                 (remove node)

                remres
                 (remove resident module)
```

```
                remstruct
                 (remove structure definition)

                remtag
                 (remove memory type definition)

                remvar
                 (remove variable)

                reqload
                 (show load requester)

                reqsave
                 (show save requester)

                request
                 (show requester)

                resident
                 (make a ML-script resident)
resm     (go to the resident module list)
reso     (go to the resource list)

                rwin
                   (open/close Refresh logical window)

                rx
                   (start ARexx script)

                save
                   (save memory to file)

                saveconfig
                 (save current PowerVisor configuration)

                savetags
                 (save memory type definitions)

                scan
                   (ask string to user)

                screen
                 (set PowerVisor on other screen)

                script
                 (start a PowerVisor or ML-script)

                scroll
                 (scroll in logical window)
scrs     (go to the screen list)

                search
                 (search a string in memory)
sema     (go to the semaphore list)

                setflags
```

```
                (set logical window flags)

               setfont
                (set font for logical window)

               showalloc
                (show all allocated memory)

               size
                  (change size for physical window)

               source
                (control source for current debug task)

               specregs
                (show special 68020 registers)

               speek
                  (peek supervisor)

               spoke
                  (poke supervisor)

               sprint
                (print on serial terminal)

               stack
                  (enable stack checking for program)

               string
                (ARexx: return string)
stru     (go to the structure definition list)

               swin
                  (open/close source logical window)

               symbol
                (control symbols for current debug task)

               sync
                  (ARexx: synchronize PowerVisor to ARexx)

               tags
                  (show all memory types currently defined)
task     (go to the task and process list)

               taskpri
                (change priority for task or process)

               tg
                  (execute command with other current tag list)

               to
                  (redirect output from command to file)

               trace
                  (control tracing in current debug task)
```

```
                  track
                     (control resource tracking)

                  unalias
                   (remove alias definition)

                  unasm
                     (disassemble memory)

                  unfreeze
                   (unfreeze a frozen task or process)

                  unhide
                   (ARexx: undo Hide command)

                  unloadfd
                   (unload a fd-file)

                  unlock
                   (unlock a lock)

                  unresident
                   (remove a resident file)

                  usetag
                   (set current tag list)

                  vars
                     (show all variables and functions)

                  view
                     (view memory in the correct type)

                  void
                     (evaluate arguments and do nothing)

                  wblock
                   (write block with trackdisk.device)
   wins     (go to window list)

                  with
                     (execute command with other current debug task)

                  xwin
                     (open/close Extra logical window)

Various:
   Back to main contents
```

1.2 Command Reference : Introduction

```
Note that some commands have a return value. When this is the case the
syntax is the following :
   <returncode> <- COMMAND <arguments> ...
```

```
Note that this is NOT the way to use the command in PowerVisor. To use
the returncode of a command you must use the group operator :
   < retcode={COMMAND <arguments> ...} <enter>


Some commands also return something in the 'rc' variable. The syntax is
as follows :
   RC,<returncode> <- COMMAND <arguments> ...


Or if something is returned in the 'input' variable :
   INPUT,<returncode> <- COMMAND <arguments> ...


In all other cases the syntax is as follows :
   COMMAND <arguments> ...


COMMAND is a sequence of characters. All uppercase characters represent
the minimum sequence you need before PowerVisor recognizes the command.
If you want to abbreviate a command, you must at least use the uppercase
characters.
```

1.3 Command Reference : All commands sorted by function

```
MMU and 68020 commands:

 mmutree

 mmureset

 mmuregs

 mmurtest

 mmuwtest

 speek

 spoke

 specregs
Information commands:

 list

 info

 llist

 gadgets

 regs
```

```
 fregs

 float

 libfunc

 libinfo

 memtask

 memory

 unasm
Logical window commands:

 colrow

 fit

 active

 current

 on

 setflags

 scroll

 setfont

 xwin

 rwin

 dwin

 awin

 owin

 swin

 openlw

 closelw
Physical window commands:

 size

 move

 openpw

 closepw
General screen commands:
```

```
 cls

 locate

 home

 color

 screen
Memory commands:

 clear

 copy

 fill

 search

 next
Special monitor commands:

 crash

 stack

 account

 track

 addfunc
~
 remfunc
Special commands:

 kill

 freeze

 unfreeze

 taskpri

 curdir

 hunks

 owner

 pathname

 opendev

 closedev

 devinfo
```

```
 devcmd

 unlock

 closewindow

 closescreen

 remove

 remhand

 remres

 remcrash

 load

 save

 loadfd

 unloadfd

 rblock

 wblock

 sprint

 event

 go

 pvcall

 for
Script commands:

 script

 rx

 resident

 unresident
ARexx commands:

 sync

 async

 front

 assign

 string
```

```
 hide

 unhide

 clip

 remclip
Debug commands:

 debug

 duse

 with

 break

 trace

 dscroll

 dstart

 drefresh

 symbol

 source
Tag and structure system:

 view

 addtag

 loadtags

 savetags

 cleartags

 remtag

 tags

 checktag

 interprete

 usetag

 tg

 addstruct

 remstruct
General commands:
```

```
                quit

                help

                void

                disp

                print

                saveconfig

                prefs

                mode

                vars

                remvar

                alias

                unalias

                attach

                remattach

                hold

                led

                showalloc

                cleanup

                request

                reqload

                reqsave

                getstring

                scan

                refresh

                error

                log

                to
              List commands:
  exec   intb   task   libs   devs   reso   memr   intr   port
  wins   attc   graf   conf   scrs   font   dosd   func   sema
```

```
   resm   fils   lock   ihan   fdfi   crsh   dbug   moni   pubs
   stru   lwin   pwin
```

1.4 Command Reference : account

```
                  ACCount


This command enables/disables stack checking and task accounting.
Default is disabled.
Use the
                 prefs
                 command to install another warning level for the
stack overflow checker.

Stack checking is done in the Exec Switch function. This means that the
stack checking resolution is not very high (see the
                 stack
                 command).

Note that the 'stack' command provides a better stack checker for one
specific task.


Related commands:
                 prefs

                 stack
```

1.5 Command Reference : active

```
                  Active <logical window>


Make the logical window the active one. The active logical window is the
logical window where you can scroll with the keys. You can also use
the <tab> key to change the active window. The active logical window is
also used for input. This means that when a command is waiting for a key
or for input (using the  key()  function, the
                 scan
                 command or simply
because the output was to big to fit on one page) input will be locked
(prevented) unless the active logical window is equal to the logical window
where the input occured.

'active' uses autodefault to the 'lwin' list for the first argument.


Related commands:
                 current
                Related functions:  getlwin()
```

```
Related lists:  lwin

Related tutor chapters:  Screens and windows
```

1.6 Command Reference : addfunc

```
                    RC,<func node> <- ADdfunc (<libfuncname> | 'offs' <library> < ↩
                        offset>)
                      ['only' <task>] [<type> [<command>]]


This is a very powerfull command which enables you to monitor
library functions. I will explain this command with examples.

'addfunc' uses autodefault to the 'task' list for the <task> argument.
'addfunc' uses autodefault to the 'libs' list for the <library> argument.

The following <type>s are available :

   .none. (do not specify) Only remember the information for the
                           8 last tasks using this function
   LED                     Blink the powerled everytime the function
                           is used. Also remember the 8 last tasks
                           using the function
   FULL                    Remember the 8 last tasks using this function
                           and also remember the registers they used to
                           call this function
   FULLLED                 Combination of FULL and LED
   EXEC                    Execute <command> everytime the function is
                           called
   SCRATCH                 Destroy the contents of the scratch registers
                           'd1', 'a0' and 'a1'. They will contain
                           $BADBADD1, $BADBADA0 and $BADBADA1 respectively

Examples :

< loadfd exec fd:exec_lib.fd <enter>
> ...

< addfunc openlibrary <enter>

Whenever some task uses the openlibrary function call, the usage count
increments and the information for the last 8 tasks using this function is
updated.

< addfunc putmsg led <enter>

Whenever a task uses the putmsg function the powerled switches its state.
The information about the 8 last tasks using this function is updated.

< addfunc wait only trackdisk.device <enter>

Everytime 'trackdisk.device' uses the Wait function, the usage count is
incremented and the information for the 8 last tasks is updated.
```

```
You can find the usage counter in the list 'func'. Type 'func' to set
this list to the current one and type 'list'.

< l func <enter>
> Function monitor    : Node     Library  Offset Traptask    Count Type
> ------------------------------------------------------------------------
> wait                : 07EA2250 07E007D8    318 07E0C6E4        2 LED
> putmsg              : 07EA1F48 07E007D8    366 00000000      314 LED
> openlibrary         : 07EA1C40 07E007D8    552 00000000       92 NORM

< remfunc wait <enter>
< remfunc putmsg <enter>
< remfunc openlibrary <enter>

< addfunc availmem full <enter>

With the
                info
                command you can see the 8 last tasks using this function
and the contents of the registers before the function is called :

< info func:availmem func <enter>
> Function monitor    : Node     Library  Offset Traptask    Count Type
> ------------------------------------------------------------------------
> availmem            : 07ED0680 07E007D8    216 00000000       13 FULL
>
> Workbench           : 07E339D0 01  07E35176    6000 Wait Workbench  (02) -
> D0: 001974E0   D1: 00000004   D2: 80000000   D3: 00000000
> D4: 00000000   D5: 00000001   D6: 40000000   D7: 80000000
> A0: 07E321B4   A1: 07E0091E   A2: 07E321B4   A3: 07E376F6
> A4: 07E326B8   A5: 07E351A8
> Workbench           : 07E339D0 01  07E35176    6000 Wait Workbench  (02) -
> D0: 00000000   D1: 00000000   D2: 80000000   D3: 00000000
> D4: 00000000   D5: 00000001   D6: 40000000   D7: 80000000
> A0: 07E321B4   A1: 07E321B4   A2: 07E321B4   A3: 07E376F6
> A4: 07E326B8   A5: 07E351A8
> Workbench           : 07E339D0 01  07E35176    6000 Wait Workbench  (02) -
> D0: 00243E50   D1: 00000002   D2: 80000000   D3: 00000000
> D4: 00000000   D5: 00000001   D6: 40000000   D7: 80000000
> A0: 07E321B4   A1: 07E0091E   A2: 07E321B4   A3: 07E376F6
> A4: 07E326B8   A5: 07E351A8
> Workbench           : 07E339D0 01  07E35176    6000 Wait Workbench  (02) -
> D0: 001974E0   D1: 00000004   D2: 80000000   D3: 00000000
> D4: 00000000   D5: 00000001   D6: 40000000   D7: 80000000
> A0: 07E321B4   A1: 07E0091E   A2: 07E321B4   A3: 07E376F6
> A4: 07E326B8   A5: 07E351A8
> Workbench           : 07E339D0 01  07E35176    6000 Wait Workbench  (02) -
> D0: 00000408   D1: 00020001   D2: 01FADB81   D3: 00000000
> D4: 40000000   D5: 80000000   D6: 00000000   D7: 00000408
> A0: 00000000   A1: 07E1EF8C   A2: 07ECF5A8   A3: 07E1EF8C
> A4: 07E220C0   A5: 07E21E58
> Workbench           : 07E339D0 01  07E35176    6000 Wait Workbench  (02) -
> D0: 001974E0   D1: 00000004   D2: 80000000   D3: 00000000
> D4: 00000000   D5: 00000001   D6: 40000000   D7: 80000000
> A0: 07E321B4   A1: 07E0091E   A2: 07E321B4   A3: 07E376F6
> A4: 07E326B8   A5: 07E351A8
> Workbench           : 07E339D0 01  07E35176    6000 Wait Workbench  (02) -
```

```
> D0: 00000000   D1: 00000000   D2: 80000000   D3: 00000000
> D4: 00000000   D5: 00000001   D6: 40000000   D7: 80000000
> A0: 07E321B4   A1: 07E321B4   A2: 07E321B4   A3: 07E376F6
> A4: 07E326B8   A5: 07E351A8
> Workbench           : 07E339D0 01  07E35176    6000 Rdy  Workbench  (02) -
> D0: 00243E70   D1: 00000002   D2: 80000000   D3: 00000000
> D4: 00000000   D5: 00000001   D6: 40000000   D7: 80000000
> A0: 07E321B4   A1: 07E0091E   A2: 07E321B4   A3: 07E376F6
> A4: 07E326B8   A5: 07E351A8

< remfunc availmem <enter>

< addfunc availmem exec print 'avail\0a' <enter>
> avail
> ...

< remfunc availmem <enter>
```

Everytime the 'availmem' Exec function is called, PowerVisor will print a message on the PowerVisor screen. You can use very complex instructions. When the command is executed you can make use of the 'rc' variable. This variable will contain the pointer to a copy of the 'func' node for the function. See the The wizard corner for more information about function nodes. Here is an example :

```
< addfunc availmem exec disp *(rc+24) <enter>
> 00000001 , 1
> 00000002 , 2
> ...

< remfunc availmem <enter>
```

When the command 'disp *(rc+24)' is executed (thus after 'AvailMem' is called) 'rc' points to a copy of the function node. At offset 24 we find the usagecounter.

Related commands:
remfunc
Related lists: func

1.7 Command Reference : addstruct

ADDStruct <filename>

Add the structure definitions from the file to the PowerVisor 'stru' list. You can then use these structures with the
interprete
and

view
commands and the peek and apeek functions.

You can look in the 'stru' list to see which structures are allready

```
in memory.

The file must be a PVSD file. This is a file made by the 'MStruct' utility
which can be found on the PowerVisor disk. 'addstruct' will complain when
the format of the file is not right.


Example :

   See the 'interprete' command


Related commands:
                 remstruct

                 interprete

                 view
                Related functions:  peek()   apeek()   stsize()

Related lists:  stru

Related tutor chapters:  Looking at things
```

1.8 Command Reference : addtag

```
                  ADDTag <address> <bytes> <type> [<structure>]


This command adds a tag to the current tag list (set with
                 usetag
                ).
With this command you can define the view mode to be used for a range of
memory. The
                 view
                 command uses these tags to determine how to dump
memory.
The <address> is the starting address for the memory range. This range
continuous for <bytes> bytes. <type> can be one of the following :

   BA       byte/ascii format
   WA       word/ascii format
   LA       long/ascii format (default for all memory not in tags)
   AS       ascii only format
   CO       code format
   ST       structure format (format supported by
                 addstruct
                )

(case is not important).
Other types can be supported in future.

'addtag' uses autodefault to the 'stru' list for the fourth argument.

If your type is 'ST' you must supply an extra argument <structure>.
```

```
This is the pointer to the structure definition you must have loaded
with the 'addstruct' command. You can use the  stsize()  function to
obtain the size of a structure. All structure reside in the 'stru' list.

Note that tags are grouped in tag lists. There are 16 tag lists numbered
from 0 through 15. The default tag list is number 0. You can use the

                 usetag
                 or
                 tg
                 command to switch to another tag list. All the tag
commands (including
                 view
                ) only work on the current tag list.


Example :

   see the
                 view
                 command.


Related commands:
                 view

                 remtag

                 cleartags

                 loadtags

                 savetags

                 tg

                 usetag

                 checktag

                 tags

                 addstruct

                 interprete
                Related functions:  taglist()   stsize()

Related lists:  stru

Related tutor chapters:  Looking at things
```

1.9 Command Reference : alias

```
                  ALias [<alias command name> <alias string>]
```

Alias without arguments lists all current aliases.

You can set a new alias <alias command name> to <alias string> with this command. You can remove the alias string with the
unalias
command.
When you create a new alias with the same name as an existing alias, the old alias will be removed.

<alias command name> is the new command that you want to define. You can't abbreviate this command like all normal commands. If you want to do this you must define another alias for that.

<alias string> is the string that is executed whenever you use <alias command name>. This will be the exact commandline except for the transformations that are described below.

Before the string is executed all [] operators in the <alias string> are replaced with the string starting after the first space after the command (the command is <alias command name>).
All other spaces before and after the commandline are not ignored and are included in [].

[<x>] with <x> one digit, is replaced with argument number <x>. The first argument is [1]. If the argument does not exist, an empty string is returned. Note that both leading and trailing spaces ARE ignored when you use this operator.

You can mix both [] and [<x>] and use them as many times as you wish.

When you want to use quotes you must precede them with '\'.

Note that alias expansion may lead to big commandlines (especially if you use recursive aliases). If the commandline gets too big you will get an error. You can increase the maximum commandline lenght with the
prefs
command.

Alias expansion is only done once. This means that you can't recursively define aliases in this manner.

When you use the group operator {}, alias expansion is done again. So you can define recursive aliases using groups.

Example :

< alias lfd 'loadfd [] fd:[]_lib.fd' <enter>

When you type 'lfd exec' on the command line, the command line will be expanded to 'loadfd exec fd:exec_lib.fd' before execution.

```
< alias execute li[]bs <enter>

When you type 'execute st li' on the command line, the command line will
be expanded to 'list libs'.
(This example is of course completely useles).


< alias writeln 'print \'[]\0a\'' <enter>

'writeln we are testing' will result in
'print 'we are testing\0a''

Look how the newline is quoted. This is because we don't want a real
newline in the alias string. We want '\0a' in the alias string.


The following will NOT create an infinite loop:

< alias disp 'list []' <enter>
< alias list 'disp []' <enter>

This sequence of command simply swappes the two commands 'disp' and 'list'.


The following three commands will generate an infinite loop :

< alias disp '{list []}' <enter>
< alias list '{disp []}' <enter>
< disp 3 <enter>
> A stack overflow was getting close !

PowerVisor stops the recursion when there is no more available stack.


The following two commands define a new command 'fact'
to compute the facultaty of its argument :

< alias _fact 'void if(([])==1,1,{_fact ([])-1}*([]))' <enter>
< alias fact 'disp {_fact []}' <enter>

< fact 5 <enter>
> 00000078 , 120

'_fact' is the recursive alias. 'fact' is only provided to give a more
command like syntax. Note that this recursion is limited by both
the available stack and the maximum length of the commandline.


You can redefine a command :

< alias disp 'disp ([])+1' <enter>

< disp 4 <enter>
> 00000005 , 5
< d 4 <enter>
> 00000004 , 4
```

```
When you use '[]' in the alias string, this sequence is replaced by
everything after the first space. For example :

< alias pr 'print \'a[]b\0a\' <enter>
< pr test <enter>
> atestb

< pr  test <enter>
> a testb


Look at the following example if you want more precise argument control :

< alias pr 'print \'Second : [2], First : [1]\0a\'' <enter>
< pr a b <enter>
> Second : b, First : a

< pr      1th      2nd    <enter>
> Second : 2nd, First : 1th


Related commands:
                 unalias
                Related tutor chapters:  Installing PowerVisor   Alias Reference
```

1.10 Command Reference : assign

```
                   ASsign <assignment string>


Assign a value to a PowerVisor variable.
This command also works for memory assignments.


Example:

   The following ARexx script (type in and execute with 'rx')

      /* Script */
      address rexx_powervisor
      a=1
      assign 'a=2'
      disp a
      disp 'a'

   Will have as output :

      < rx file <enter>
      > 00000001 , 1
      > 00000002 , 2


Related commands:
                 rx
```

```
                vars

                remvar

                script

                assign

                error

                clip

                remclip

                string
               Related tutor chapters:  Scripts
```

1.11 Command Reference : async

```
                  ASYnc


Use this command to disable the synchronization enabled with
                sync
               .


Related commands:
                sync

                rx
               Related tutor chapters:  Scripts
```

1.12 Command Reference : attach

```
                  <attach node> <- ATtach <command string> <code> <qualifier>
                   [('e' | 'c') ['+']]


This command attaches a command to a key. You can use this command to
initialize your functionkeys (such a command attached to a key is
called a macro).

With
                rematTach
                you can remove macros.

When you press the defined key, the command will be copied to the
commandline. The effect is exactly the same as it would be if you
had typed in the command and pressed enter.
```

If you do not want the command to be feedbacked on your screen if you
press the assigned key, you may want to precede the commandline with a '~'.

If you use the 'e' (Exec) option, the command is executed without
disturbing the commandline. The command is not feedbacked on the screen.

If you use the 'c' (Copy) option, the <command string> is copied to the
current position in the stringgadget. Nothing is executed.

If you use either the 'e' or 'c' options, you can add a '+' to indicate
that the key should not be filtered out by the input handler. This means
that the pressed key is handled normally. After the default behaviour,
the command is executed ('e') or the command string is copied to the
commandline ('c').

Example :

To add the
                list
                command to the <F1> key you can use :

< attach 'list' 050 0 <enter>

If you press <F1> 'list' is put on the stringgadget and an enter is
simulated (this means that 'list' is also put in the history buffer).

If you do not want the stringgadget to be disturbed you can use :

< attach 'list' 050 0 e <enter>

Related commands:
                rematttach
               Related lists: attc

Related tutor chapters: Installing PowerVisor   Scripts

1.13 Command Reference : awin

                  AWin [<number of lines>]

Open/close the Rexx window. When the 'Rexx' logical window is open,
PowerVisor will use it for Rexx output. Otherwise the current logical
window is the one that will receive the output.

Normally the height of the new logical window is 30 % of the total
physical window height ('Main' is the physical window for all standard
logical windows). However, if you specify <number of lines>, the logical
window will be opened with <number of lines> visible lines (This is
the number of lines when the default PowerVisor font is used).

By default the 'Rexx' logical window has the following characteristics :

```
   - Number of columns is fixed and equal to the maximum number of columns
     visible at the time the logical window is created
   - Number of rows is fixed and is always equal to 50
   - -MORE- checking is disabled
   - Interrupt/Pause checking is disabled
   - Home position is top-visible
   - Auto Output Snap is off

You can change these characteristics with the
                 prefs
                 command.

Note that if the 'intui' mode flag is one (this is off by default, see the

                 mode
                 command) the logical window will be opened on a new physical  ↩
                    window.
<number of lines> is ignored in that case.


Related commands:
                 rwin

                 dwin

                 swin

                 xwin

                 owin

                 fit

                 colrow

                 prefs

                 mode
                Related lists:  lwin

Related tutor chapters:  Screens and windows   Scripts
```

1.14 Command Reference : break

```
                  RC <- Break ('a'|'c'|'n'|'t'|'p'|'r') <address> [<condition>|< ↩
                     timeout>]


Install a breakpoint for the current debugtask.

   break a <address> <timeout>   Breakpoint only breaks after the <timeout>
                                 counter becomes zero.
   break c <address> <condition> Only break if the condition is true.
   break n <address>             Normal breakpoint.
   break t <address>             Temporary breakpoint.
```

```
   break p <address>             Profiler breakpoint. Does not break. Only
                                 increments a counter.
   break r <breakpoint number>   Remove a breakpoint.

'rc' contains the pointer to the new breakpoint node.


Examples :

< break c 00c05320 '(@d0<@d1)&&(@pc>00c05380)'

This breakpoint will cause a break if d0 is lower than d1 and
the programcounter is greater than 00c05380.


Related commands:
                 debug

                 trace

                 duse

                 with

                 symbol
                Related functions:  getdebug()

Related lists:  dbug

Related tutor chapters:  Debugging
```

1.15 Command Reference : checktag

```
                  <bytes remaining> <- CHecktag <address>


Check if an address resides in the current tag list. If it is, PowerVisor
will print the type of the tag list followed by the remaining number of
bytes after <address> in this tag.
If the address is not in the tag list, PowerVisor will print the number
of bytes remaining before the next tag. If there is no next tag, PowerVisor
will print $7fffffff.


Example :

< addtag 1000 50 as <enter>

< checktag 1010 <enter>
> AS 00000028,40

< checktag 900 <enter>
> 00000064,100

< checktag 1100 <enter>
```

```
> 7FFFFFFF,2147483647


Related commands:
                  addtag

                  remtag

                  tags

                  cleartags

                  loadtags

                  savetags

                  usetag

                  tg

                  view
                 Related functions:  taglist()

Related tutor chapters:  Looking at things
```

1.16 Command Reference : cleanup

```
                    CLEANup


This command frees all memory blocks that are allocated with the
 alloc()  function and the
                  rblock
                  command. This is also
done automatically when PowerVisor quits.


Related commands:
                  showalloc

                  rblock
                 Related functions:  alloc()   free()   getsize()   isalloc()    ↩
                    realloc()

Related tutor chapters:  Scripts
```

1.17 Command Reference : clear

```
   CLEar [<value>]


Fill all unused memory with <value> (or 0 if value is not specified).
```

<value> is a longword.

Example :

< clear $ABCDEF00 <enter>

This command will fill all free memory with $ABCDEF00. So you can easily
see which memory is free and which memory is not if you view the memory.

1.18 Command Reference : cleartags

CLEARTags

Clear all tags in the current tag list.

Related commands:
remtag

addtag

loadtags

savetags

tags

usetag

tg

checktag

view
Related functions: taglist()

Related tutor chapters: Looking at things

1.19 Command Reference : clip

[RC,<Pointer to data>] <- CLIp <Clip name> [<Pointer to data> < ↩
Length>]

If you supply the <Pointer to data> and <Length> arguments this command
can be used to install a new clip in the ARexx system. In this case the
return value (RC) is undefined.

If you don't supply these two extra arguments, PowerVisor will search the
clip and return the pointer to the data in 'RC' (or 0 if not found). Note

```
that this pointer is actually a pointer to the string part of the 'RexxArg'
structure so you can find the length of the data in *(RC-4).w (the word
four bytes before the data).
The pointer is also displayed on screen.


Example:


< clip TEST "Testing 1 2 3..." 16 <enter>

< memory {-clip TEST} 16 <enter>
> 07E9A0F0: 54657374 696E6720 31203220 332E2E2E              Testing 1 2 3...

CLI< rx 'say getclip(test)' <enter>
CLI> Testing 1 2 3...


Related commands:
                 remclip

                 rx

                 assign
                Related tutor chapters :  Scripts
```

1.20 Command Reference : closedev

```
                   CLOSEDev <device block>


Close a device opened with
                 opendev
                .


Related commands:
                 opendev

                 devcmd

                 devinfo
```

1.21 Command Reference : closelw

```
                   CLOSELw <logical window>


Close a logical window previously opened with
                 openlw
                 or one of the
standard logical window commands (xwin,rwin,awin,owin,dwin,swin).
```

```
You can't close the 'Main' logical window.

'closelw' uses autodefault to the 'lwin' list for the first argument.


Related commands:
                 openlw

                 awin

                 owin

                 rwin

                 dwin

                 swin

                 xwin

                 openpw

                 closepw
               Related lists:  lwin

Related tutor chapters:  Screens and windows
```

1.22 Command Reference : closepw

```
                  CLOSEPw <physical window>


Close a physical window. All logical windows on this physical window
are automatically removed.
You can't remove the 'Main' physical window.

'closepw' uses autodefault to the 'pwin' list for the first argument.


Related commands:
                 openpw

                 openlw

                 closelw
               Related lists:  pwin

Related tutor chapters:  Screens and windows
```

1.23 Command Reference : closescreen

```
                    CLOSEScreen <screen>


This command closes a screen. All windows on this screen are also closed.

'closescreen' uses autodefault to the 'scrs' list for the first argument.

Be careful with this command. Closing screens that do not belong to you
is not so friendly for the other program.


Related commands:
                  closewindow
                 Related lists:  scrs
```

1.24 Command Reference : closewindow

```
                    CLOsewindow <window>


This command closes a window. The menu's are cleared. The DMRequest is
cleared. All requesters attached to this window are removed and the IDCMP
flags are set to zero.

'closewindow' uses autodefault to the 'wins' list for the first argument.

Be careful with this command. Closing windows that do not belong to you
is not so friendly for the other program (of course you can simply kill
or freeze the other command and you will hear no complaints :-)


Related commands:
                  closescreen
                 Related lists:  wins
```

1.25 Command Reference : cls

```
                    CLs


This command clears the current logical window.
After the window is cleared the current position is set to the home
position of the logical window. This home position can be on two
different places :

   - the real (0,0) home position. Logical windows like 'Debug' and
     'Refresh' have this position as their home position. Such windows are
     called real-top windows. When such a window is cleared, PowerVisor
     automatically scrolls to the (0,0) position.
   - the top position of the bottom visible part of the logical window.
     Logical windows like 'Main' have this position as their home
```

```
     position. These windows are called top-visible windows. When such
     a window is cleared, PowerVisor scrolls to the bottom visible part
     of the logical window and sets the current position to the top
     line of this visible part.

Note that you can change the behaviour of each logical window with the

                 prefs
                 command.


Related commands:
                 current

                 prefs
                Related functions:  getlwin()

Related lists:  lwin

Related tutor chapters:  Screens and windows
```

1.26 Command Reference : color

```
                   COLor <col num> <red> <green> <blue>


Change the rgb value of a PowerVisor screen colour.
Only use 0 or 1 for <col num> (except if you have a two bitplane screen).

This command gives an error when you try to use it when
PowerVisor is on another screen.


Related commands:
                 screen
                Related tutor chapters:  Screens and windows
```

1.27 Command Reference : colrow

```
                   COLRow <logical window> <columns> <rows>


Set the number of columns and rows (in characters) for the specified
logical window. If <columns> or <rows> are equal to -1, PowerVisor will
scale them automatically to the maximum size available. The disadvantage
of this is that the logical window is cleared everytime the visible size
(in the direction that is -1) changes.
The number of columns and rows can be as big as you want. If the size is
to big you can scroll in the logical window using the following keys
(in combination with the Left-Alt key) on the numeric keypad :

   8 (arrow up)      scroll logical window one line up
```

```
   2 (arrow down)    scroll one line down
   6 (arrow right)   scroll one column right
   4 (arrow left)    scroll one column left
   9 (PgUp)          scroll 5 lines up
   3 (PgDn)          scroll 5 lines down
   7 (Home)          scroll to the top left position
   1 (End)           scroll to the bottom left position
   5                 scroll to the right as far as you can
```

If there are more logical windows open on the screen you can choose which one you want to scroll by pressing the <Tab> key. The active window is the logical window with the full (blue for two bitplanes or black for one bitplane) status bar. The other logical windows have an empty status bar. Note that the active logical window is not the same as the current logical window. The active logical window is only used to indicate in which window you can scroll with the keys. The current logical window is the window that will get the output for all commands. All input is also redirected to the active logical window (see the Screens and Windows chapter for more info).

'colrow' uses autodefault to the 'lwin' list for the first argument.

See the 'lwin' list for all available logical windows.
There are seven standard logical windows :

```
   Rexx     : ARexx output window
   PPrint   : PortPrint output window
   Refresh  : refresh window
   Debug    : debugging window
   Extra    : extra window
   Main     : main window
   Source   : source window for source level debugger
```

You can use the getcol() and getrow() functions to ask the number of rows and columns respectively that you have set with this command. If you want the real number of rows and columns (after scaling) you can use the cols() and lines() functions.

Note that you can change the <tab> key to any other key with the

```
                prefs
                command.
```

Example :

Make the 'Main' logical window autoscalable in both directions :

```
< colrow main -1 -1 <enter>
```

You can't scroll in this logical window now, and everytime the visible size changes the window is cleared (since there is a new rescaling).

Make the 'Main' logical window very big in both directions :

```
< colrow main 200 100 <enter>

(200 columns and 100 rows). Type some output on this window :

< help commands <enter>
> ...

Now you can scroll using the keys explained above.

You can use the
                 fit
                 command to scale the size back to normal.


Related commands:
                 fit

                 xwin

                 rwin

                 dwin

                 swin

                 awin

                 owin

                 openlw

                 closelw

                 prefs
                Related functions:  getcol()   getrow()   cols()   lines()

Related lists:  lwin

Related tutor chapters:  Screens and windows
```

1.28 Command Reference : copy

```
                  Copy <source> <destination> <bytes>


Copy <bytes> bytes from <source> to <destination>.
Be cautious with this command. It can be very destructive.


Related commands:
                 fill

                 search
```

1.29 Command Reference : crash

```
                    CRAsh <taskpointer>
```

This command patches the TrapCode in the task to the PowerVisor trapcode routine. This means that PowerVisor will trap all exceptions for that task. This command is useful in combination with 'mode patch'. 'mode patch' patches all new tasks (by patching AddTask) while 'crash' patches all existing tasks (to patch all tasks use 0 for the <taskpointer>). You should use this command if you want to debug using resident breakpoints or if you want to be able to debug a task after it crashes (in that case you should have done this command BEFORE the task crashes).

Note that all these difficulties are only for tasks already running when PowerVisor was started (provided 'mode patch' is true). All new tasks will get the proper attention of PowerVisor.

Note that it is sometimes dangerous to patch all tasks at once since it is possible that a program uses the TrapCode field for other purposes. If you have such programs use 'crash' with a specific task node.

'crash' uses autodefault for the 'task' list for the first argument.

Related commands:
mode

1.30 Command Reference : curdir

```
   CUrdir <process> <directory name>
```

Set a new current directory for a process.

'curdir' uses autodefault to the 'task' list for the first argument.

Example :

```
< curdir Shell :langua/sources <enter>
```

Related lists: task

1.31 Command Reference : current

```
                    CURRent <logical window>
```

Set the current logical window. Only 'Extra' and 'Main' are allowed. Before you can use the 'extra' logical window you must open it with

```
the
                  xwin
                  command or the
                  openlw
                  command.

'current' uses autodefault to the 'lwin' list for the first argument.

You can use the  getlwin()  function to see which logical window is
current. Use the
                  on
                  command to temporarily set the current logical window
(for one command).


Example :

Open 'Extra' logical window (if not already open) :

< xwin <enter>

< current extra <enter>
< l task <enter>
> ...

Output appears on 'Extra' logical window.

Close 'Extra' window :

< xwin <enter>


Related commands:
                  xwin

                  on

                  openlw
                 Related functions:  getlwin()

Related lists:  lwin

Related tutor chapters:  Screens and windows
```

1.32 Command Reference : debug

```
                    RC <- DEBug 'n' | 'c' | 'l' <filename> | 't' <task node> | ('r ↩
                       '|'f'|'u')
          [<debug node>] | 'd' <name>


Use this command to control the debug tasks.

   debug n      Waits for the next 'LoadSeg' and make a debugtask from that
                process.
```

```
   debug c     Waits for the next 'AddTask' and make a debugtask from that
               task.
   debug l <filename>
               Load an executable program to debug.
               Symbols are automatically loaded if they exist.
               Warning ! If you use the AmigaDOS 1.3 version, PowerVisor
               will NOT create a CLI for the loaded program. Therefore
               it is better to use 'debug n' if possible. In AmigaDOS 2.0
               the CLI structure is properly allocated (using
               'CreateNewProc')
   debug t <task node>
               Take an existing running task to debug.
               You can also take a crash node to debug. In that case
               you can singlestep the task from where it crashed
   debug f     Remove the current debugnode and freeze the task
   debug f <debug node>
               Remove the specified debugnode and freeze
               This command is mostly used when 'debug n' or 'debug t' was
               used to start the debugging
   debug r     Remove the current debugnode.
               The task will simply continue executing from where it
               was interrupted.
               This command is mostly used when 'debug n', 'debug c' or
               'debug t' was used to start the debugging
               If you have AmigaDOS 2.0 you may also use 'debug r' with
               'debug l'
   debug r <debug node>
               Remove the specified debugnode.
   debug u     Remove the current debugnode.
               The task will be stopped ('RemTask') and the program will
               be unloaded (UnLoadSeg) if the debug task was loaded with
               'debug l'. This command is mostly used when 'debug l' was
               used
               If you have AmigaDOS 2.0 you may also use 'debug u' with
               'debug n'
   debug u <debug node>
               Remove the specified debugnode.
   debug d <name>
               Create dummy debugnode for symbols only.

When a new debug task is created, it is put in the 'DBug' list.
'rc' contains the pointer to the new debug node.

'debug' uses autodefault to the 'dbug' list for the <debug node> argument.
'debug' uses autodefault to the 'task' list for the <task node> argument.


Related commands:
                duse

                trace

                symbol

                break

                source
```

```
                Related functions:  getdebug()

Related list:  dbug   task

Related tutor chapters:  Debugging
```

1.33 Command Reference : devcmd

```
                  RC <- DEVCmd <dev> <command> [<flags> [<length> [<data> [< ↩
                     offset>]]]]


Give a command to a device opened with
                 opendev
                . The parameters specified
are for the IORequest. If you do not specify these parameters ('flags',...)
the IORequest will remain unchanged. The variable 'rc' contains the result
of the 'DoIO' function.


Related commands:
                 opendev

                 closedev

                 devinfo
```

1.34 Command Reference : devinfo

```
                  DEVInfo <device block>


Show port and iorequest information for a device opened with 'opendev'.


Related commands:
                 opendev

                 closedev

                 devcmd
```

1.35 Command Reference : disp

```
                  <value of expression> <- Disp <expression>


Use this command to display an expression.
```

```
Example :


< disp 4*(7-3) <enter>
> 00000010 , 16


Related commands:
                 print

                 locate
                Related tutor chapters:  Expressions
```

1.36 Command Reference : drefresh

```
                   DRefresh


This command refreshes the debug display. 'drefresh' is only useful when
you are in fullscreen debugging mode.

See the
                 prefs
                ,
                 dwin
                 and
                 swin
                 commands to install the fullscreen
debugger.


Related commands:
                 dwin

                 swin

                 debug

                 duse

                 prefs
                Related lists:  dbug

Related tutor chapters:  Debugging
```

1.37 Command Reference : dscroll

```
                   DScroll <offset>


Scroll <offset> bytes up in the fullscreen debugger. You can only use this
command when the 'Debug' logical window is open (with
```

```
                dwin
               ). <offset>
can be negative. PowerVisor will always convert <offset> to a multiple
of a word.

You can also scroll in the debug display with the 'Ctrl' key in combination
with the following keypad keys :

   8 (up)      scroll 2 bytes up
   2 (down)    scroll 2 bytes down
   9 (PgUp)    scroll 20 bytes up
   3 (PgDn)    scroll 20 bytes down
   5           scroll to programcounter (use 'dstart' to simulate this)


Related commands:
                dstart

                debug

                trace

                break

                dwin
               Related functions:  toppc()   botpc()

Related lists:  lwin   dbug

Related tutor chapters:  Debugging
```

1.38 Command Reference : dstart

```
                  DSTart <address>


Set the start of the debug logical window. You can only use this
command when the 'Debug' logical window is open (with
                dwin
               ).


Related commands:
                dscroll

                debug

                trace

                break

                dwin
               Related functions:  toppc()   botpc()

Related lists:  lwin   dbug
```

Related tutor chapters: Debugging

1.39 Command Reference : duse

DUse <debug node>

Set the current debug task.
Use the getdebug() function to determine the current debug node.

Also see the
with
command to temporarily set the current debug node.

'duse' uses autodefault to the 'dbug' list for the first argument.

Related commands:
with

debug

break

trace

drefresh

symbol
Related functions: getdebug()

Related lists: dbug

Related tutor chapters: Debugging

1.40 Command Reference : dwin

DWin [<number of lines>]

Open/close the debug window. (required if you want to use
fullscreen debugging).

Normally the height of the new logical window is 30 % of the total
physical window height ('Main' is the physical window for all standard
logical windows). However, if you specify <number of lines>, the logical
window will be opened with <number of lines> visible lines (This is
the number of lines when the default PowerVisor font is used).

By default the 'Debug' logical window has the following characteristics :

```
   - Number of columns is fixed and equal to 82
   - Number of rows is fixed and equal to 42
   - -MORE- checking is disabled
   - Interrupt/Pause checking is disabled
   - Home position is real-top
   - Auto Output Snap is off

You can change these characteristics with the
                 prefs
                 command.

You can't use
                 on
                 with the 'Debug' logical window.

Note that if the 'intui' mode flag is one (this is off by default, see the

                 mode
                 command) the logical window will be opened on a new physical  ↩
                    window.
<number of lines> is ignored in that case.


Related commands:
                 rwin

                 swin

                 xwin

                 awin

                 owin

                 debug

                 prefs

                 mode

                 on
                Related lists:  lwin

Related tutor chapters:  Screens and windows   Debugging
```

1.41 Command Reference : error

```
                  <pointer to error string> <- ERror <error number>


Get a pointer to an error string (see the  geterror()  function
for a list of error numbers). This contents of this pointer is very
volatile because the pointer actually points to a very much used area
in PowerVisor. Because of this, this command has only limited use. The
command is more useful in ARexx where it returns a string containing the
```

```
error message.


Example:


The following command results in rubbish because the
                 memory
                 command
immediatelly needs the area containing the error string.

< memory {error 17} <enter>
> <rubbish>


If you want a copy of the string you can use the  alloc()  function :

< str=alloc(s,#{error 17}) <enter>
< memory str 16 <enter>
> 07E9C25A: 4D697373 696E6720 6F706572 616E6420               Missing operand


Related commands:
                 rx

                 script
               Related functions:  geterror()

Related tutor chapters:  Scripts
```

1.42 Command Reference : event

```
                  EVent <class> <subclass> <code> <qualifier> <x> <y>


Add an input event to the input event chain. This command is most useful
in macros and in ARexx.
Note that you if you want to use an EventAddress for a specific event,
you can split this event address in two parts and supply them instead
of <x> and <y>.

Possible class values :

   NULL           0
   RAWKEY         1
   RAWMOUSE       2
   EVENT          3
   POINTERPOS     4
   TIMER          6
   GADGETDOWN     7
   GADGETUP       8
   REQUESTER      9
   MENULIST       10
   CLOSEWINDOW    11
   SIZEWINDOW     12
```

```
   REFRESHWINDOW  13
   NEWPREFS       14
   DISKREMOVED    15
   DISKINSERTED   16
   ACTIVEWINDOW   17
   INACTIVEWINDOW 18
   NEWPOINTERPOS  19    (AmigaDOS 2.0 only)
   MENUHELP       20    (AmigaDOS 2.0 only)
   CHANGEWINDOW   21    (AmigaDOS 2.0 only)

Possible subclass values :

   For class NEWPOINTERPOS :

      COMPATIBLE  0
      PIXEL       1
      TABLET      2

Possible code values :

   For class RAWKEY :

      UP_PREFIX         $80
      KEY_CODE_FIRST    $00
      KEY_CODE_LAST     $77
      COMM_CODE_FIRST   $78
      COMM_CODE_LAST    $7f

   For class RAWMOUSE :

      LBUTTON           $68
      RBUTTON           $69
      MBUTTON           $6a
      NOBUTTON          $ff

   For class EVENT (AmigaDOS 2.0 only) :

      NEWACTIVE         1
      NEWSIZE           2
      REFRESH           3

   For class REQUESTER :

      REQCLEAR          0
      REQSET            1

Possible qualifier values :

   LSHIFT         $0001
   RSHIFT         $0002
   CAPSLOCK       $0004
   CONTROL        $0008
   LALT           $0010
   RALT           $0020
   LCOMMAND       $0040
   RCOMMAND       $0080
   NUMERICPAD     $0100
```

```
   REPEAT         $0200
   INTERRUPT      $0400
   MULTIBROADCAST $0800
   MIDBUTTON      $1000
   RBUTTON        $2000
   LEFTBUTTON     $4000
   RELATIVEMOUSE  $8000


Related commands:
                 attach
```

1.43 Command Reference : fill

```
                  Fill <dest> <bytes> <with>


Fill the memory starting at <dest> with the string <with>. Do this
for <bytes> bytes.

Be cautious with this command. It can be very destructive.


Example :

< a=alloc(n,1000) <enter>
< fill a 22 'test\41' <enter>

will copy 4.4 times the string "testA" beginning at address in a.
The memory map looks something like:

< m a <enter>
> 07E79AB2: 74657374 41746573 74417465 73744174              testAtestAtestAt
> 07E79AC2: 65737441 74650000 00000000 00000000              estAte..........
> 07E79AD2: 00000000 00000000 00000000 00000000              ................
> 07E79AE2: 00000000 00000000 00000000 00000000              ................
> ...


Related commands:
                 copy

                 search
```

1.44 Command Reference : fit

```
                  FIT <logical window>


This command adjusts the number of columns and number of rows of a
logical window to fit exactly in the visible size. This is useful after a
switch from non-interlace to interlace for example. You can't scroll in the
```

```
logical window after a 'fit' since the window is just as large as the
visible size.

You can achieve the same result with the
                colrow
                command, but then you
have to compute the number of columns and rows for yourselves.

'fit' uses autodefault to the 'lwin' list for the first argument.


Example :

Open the 'Extra' logical window :

< xwin <enter>

< fit extra <enter>


Related commands:
                colrow

                xwin

                rwin

                dwin

                swin

                awin

                owin
               Related lists:  lwin

Related tutor chapters:  Screens and windows
```

1.45 Command Reference : float

```
                  FLoat <task>|<debug node> <register number> <register value>


Change the value of a floating point register in a primitive way
(At this moment only double and single format are supported and no
exponent. The number of digits left and right of the decimal point
is also limited to nine, all other digits will be ignored).

<register number> may be 0 to 7 for fp0 to fp7 respectively.


Example :

Change fp3 for the current debug task:
```

```
< float getdebug() 3 -3.1415926 <enter>


Related commands:
                 fregs
```

1.46 Command Reference : for

```
                   FOR <listname> <command>


For each element in list <listname> execute <command>.
The command is executed with the pointer to the current element from the
list in the 'rc' variable.

Warning ! Don't use commands that may cause the removal of some of the
elements in the list.


Example :

< for task disp rc <enter>
> 07E2D4B0 , 132306096
> 07E4DE50 , 132439632
> 07E5CBB0 , 132500400
> 07E1C418 , 132236312
> 07E49DB8 , 132423096
> 07E1B788 , 132233096
> 07E62768 , 132523880
> 07E1D510 , 132240656
> 07E60860 , 132515936
> 07E0AF7A , 132165498
> 07E4CD20 , 132435232
> 07E08000 , 132153344
> 07E07C68 , 132152424
> 07E70388 , 132580232
> 07EB4F48 , 132861768
> 07E0DEE0 , 132177632
> 07E13880 , 132200576
> 07E15D00 , 132209920
> 07E339D0 , 132331984
> 07E0C6E4 , 132171492
> 07E4FEE8 , 132447976
> 07E06362 , 132146018
> 07E21B28 , 132258600
> 07EB5948 , 132864328

displays all pointer to tasks.

To count all resident modules for example, use :

< {a=0;for resm a=a+1;disp a} <enter>
> 0000002A , 42
```

```
Related commands:
                 list
                Related tutor chapters:  List Reference   Looking at things
```

1.47 Command Reference : freeze

```
                   FReeze <task>


Freeze a task. Remember that when you quit PowerVisor all frozen tasks
are lost.

Use the
                 unfreeze
                 command to unfreeze a task.
In the 'task' list all frozen tasks have 'Cold' as their 'Stat'.

'freeze' uses autodefault to the 'task' list for the first argument.


Related commands:
                 kill

                 unfreeze
                Related lists:  task
```

1.48 Command Reference : fregs

```
                   FREGs <task>|<debug node>


Show the floating point registers for a task in a primitive way.
Each floating point register (fp0..fp7) is shown in the internal
Exec stackframe format (12 bytes) followed by the real value of
the register (At this moment only double and single format are supported
and no exponent. The number of digits left and right of the decimal
point is also limited to nine).

Future enhancements will provide better output and more support in
general for floating point debugging.


Example :

Show all floating point register for the current debug task:

< fregs getdebug() <enter>
> 3FF50000 83126E97 8D4FD800 : 0.000999999
> 40040000 AFFFFFFF FFFFFA00 : 43.999999999
> BEEB0000 8637BD05 AF6C6800 : -0.000000999
> 3FFF0000 9E065152 D8EAE800 : 1.234567799
> C11C0000 EE6B27FF FFFFF800 : -999999999.999999880
```

```
> BFFF0000 80000000 00000000 : -1.000000000
> 40000000 C90FDAA1 7F49B800 : 3.141592652
> 40020000 A0000000 00000000 : 10.000000000


Related commands:
                 regs

                 float
```

1.49 Command Reference : front

```
                   FROnt


Bring the PowerVisor screen to the front from within an ARexx script.


Related commands:
                 rx
               Related tutor chapters:  Scripts
```

1.50 Command Reference : gadgets

```
                   Gadgets <window>


Lists all gadgets in the specified window.

'gadgets' uses autodefault to 'wins' for the first argument.


Example :

< l wins <enter>
> Window name         : Address  Left  Top Width Height WScreen
> ------------------------------------------------------------------------
>                     : 07E42A78    0   12   692    430 07E20300
>                     : 07E8C810    0    0   704    456 07E77A60
> Clock               : 07E66BD0  558  336   120    140 07E280D0
> My Shell            : 07E414E0    0  568   692    456 07E280D0
>                     : 07E3B078    0   16   692   1008 07E280D0

< gadgets clock <enter>
> Gadget ptr : left right width height Render   Text     SpecInfo ID
>
> 07E63C2C   :  -22     0    24     16 07E43BC4 00000000 00000000     0
> Flags      : GADGHCOMP GADGIMAGE GRELRIGHT LABELITEXT
> Activation : RELVERIFY BORDERSNIFF
> Type       : SYSGADGET WUPFRONT CUSTOMGADGET
>
> 07E63C6C   :  -45     0    24     16 07E5506C 00000000 00000000     0
```

```
> Flags      : GADGHCOMP GADGIMAGE GRELRIGHT LABELITEXT
> Activation : RELVERIFY BORDERSNIFF
> Type       : SYSGADGET WDOWNBACK CUSTOMGADGET
>
> 07E6682C   :  -17    -9    18     10 07E5549C 00000000 00000000     0
> Flags      : GADGHCOMP GADGIMAGE GRELBOTTOM GRELRIGHT LABELITEXT
> Activation : RELVERIFY BORDERSNIFF
> Type       : SYSGADGET SIZING CUSTOMGADGET
>
> 07E66C84   :    0     0    20     16 07E558CC 00000000 00000000     0
> Flags      : GADGHCOMP GADGIMAGE LABELITEXT
> Activation : RELVERIFY BORDERSNIFF
> Type       : SYSGADGET CLOSE CUSTOMGADGET
>
> 07E66CC4   :    0     0     0     15 00000000 00000000 00000000     0
> Flags      : GADGHCOMP GADGIMAGE GRELWIDTH LABELITEXT
> Activation : BORDERSNIFF
> Type       : SYSGADGET WDRAGGING CUSTOMGADGET


Related commands:
                 list

                 info
                Related lists:  wins

Related tutor chapters:  Looking at things
```

1.51 Command Reference : getstring

```
                  INPUT,<ptr to inputline> <- GEtstring <title> <max number of  ↩
                     chars>


Ask the user for input. This command uses reqtools.library (V37 or
higher) (© Nico François) if available, otherwise this command is
equivalent to the
                 scan
                 command.
The result of this command (in input), is a pointer to the result string
(or 0 if the user pressed 'cancel'). This string is remembered until you
quit PowerVisor or until you use another input command (
                 scan
                ,
                 reqload
                ,

                 reqsave
                 or
                 getstring
                ).

This command returns a string if used from ARexx.

<title> is the title to be put in the requester window
```

```
<max number of chars> is the maximum number of characters that is allowed
in the requester

Both parameters are ignored if 'scan' is used


Example :


< getstring 'This is the title' 256 <enter>
Requester> This is a test <enter>

< memory input 20 <enter>
> 07EEBC82: 54686973 20697320 61207465 73740000              This is a test..
> 07EEBC92: 00000000                                        ....


Related commands:
                 scan

                 request

                 reqload

                 reqsave
                Related functions:  key()
```

1.52 Command Reference : go

```
                   <result> <- GO <address> [<commandline>]


Start executing at <address>.
You can make inline code with this command.
The registers are preset to certain values (see the
                 script
                 command).

With the
                 resident
                 command you can load a ml-script into memory. You
can use the 'go' command to execute this script.


Example :

< disp {go "\70\01Nu"} <enter>
> 00000001 , 1

Because this program is equivalent to :

      moveq.l  #1,d0
      rts
```

```
Related commands:
                 script

                 pvcall

                 resident

                 unresident
                Related tutor chapters:  Scripts
```

1.53 Command Reference : help

```
   Help [<topic>]


Ask more information about a certain topic. You must have PowerVisor-help
and PowerVisor-ctrl installed in your s: or program directory before you
can use 'help'. If you omit <topic> you will get the main menu.


Related tutor chapters:  Getting Started
```

1.54 Command Reference : hide

```
                   HIde


Hide all output from commands issued from an ARexx script.
This is equivalent to using the '-' operator in front of a commandline
(when you are typing commands from PowerVisor). But you can't type a '-'
in front of an ARexx commandline, so you have to use 'hide'.


Related commands:
                 unhide

                 rx
                Related tutor chapters:  Scripts
```

1.55 Command Reference : hold

```
                   HOld


This command closes the PowerVisor screen and waits for the Right-Alt,
Right-Shift,? combination to reopen it again.
If a crash happens PowerVisor will reopen its screen automatically.

Note that you can redefine the hot key to any key combination you want
with the
```

```
                prefs
                command.


Related commands:
                quit

                prefs
               Related tutor chapters:  Getting Started
```

1.56 Command Reference : home

```
                  HOMe


Adjust the current location on the current logical window to the
top location. Note that this is not always the same as 'locate 0,0'
(see the  Screens and windows  chapter for more info).
This command also scrolls the logical window to that position
(in contrast with
                locate
               ).


Related commands:
                locate

                cls

                print

                current
               Related functions:  getx()   gety()   getchar()   lines()   cols()
                   getlwin()

Related lists:  lwin

Related tutor chapters:  Screens and windows
```

1.57 Command Reference : hunks

```
   HUnks <process>


Show all the hunks for a process

'hunks' uses autodefault to the 'task' list for the first and only
argument.


Related lists:  task
```

Related tutor chapters: Looking at things

1.58 Command Reference : info

```
Info <object address> [<list>]
```

The 'info' command can be used to ask more information about an element of a list.
Note that the information this command gives will contain no BPTR's. These are automatically converted to APTR's.

Examples :

Let's assume we have the 'PowerVisor' task in the task list (this is in fact the case since you have probably started PowerVisor :-) :

```
< info task:powervisor task <enter>

or

< info powervisor task <enter>

or

< info powervisor <enter>

or

< info task:powervisor <enter>

> Task node name      : Node     Pri StackPtr  StackS Stat Command        Acc
> ------------------------------------------------------------------------
> PowerVisor1.0.task  : 07E70370 00  07E7137E    4096 Wait            TASK -
>
> IDNestCnt    : 00       | TDNestCnt    : FF       | SigAlloc     : E000FFFF
> SigWait      : E0000000 | SigRecvd     : 00000000 | SigExcept    : 00000000
> TrapAlloc    : 8000     | TrapAble     : 0000     | ExceptData   : 00000000
> ExceptCode   : 00F83AEC | TrapData     : 00000000 | TrapCode     : 07E77696
> SpLower      : 07E703D0 | SpUpper      : 07E713D0 | SpReg        : 07E7137E
> MemEntry     : 07E703BA | UserData     : 00000000 |
```

The first version of the command is the safest one. There is nothing that can go wrong there.

The second version could crash if the current list is not equal to the 'task' list. This is because PowerVisor will then try to interprete the element starting with 'powervisor' in the other current list as a task. (If you are lucky there is no such element in the current list, in that case you simply get an error).

The third version is also safe although you could end up with the wrong information for the wrong element in the wrong list.

```
The last version could crash if the current list is not equal to the
'task' list. It is in fact the most dangerous version of all.

The 'info' command does not work for the 'Exec', 'IntB' and 'Graf' lists.
This is because these lists are structures and already give you all
information there is.


Related commands:
                 list
                Related tutor chapters:  Looking at things
```

1.59 Command Reference : interprete

```
                   INTErprete <structure pointer> <struct def pointer>


Show the contents of a structure at <structure pointer> using the
<struct def pointer> structure definition. You can load and add
structure definitions with the
                 addstruct
                 command.

All structure definitions reside in the 'stru' list.

You can also view structures using the
                 view
                 command.

'Interprete' uses autodefault to the 'stru' list for the second argument.


Example :

Add all exec structure definitions.

< addstruct exec.pvsd <enter>
> UNIT
> IS
> IV
> ..
> SS
> SM
> TC
> ETask
> StackSwapStruct

All these structure are now added to the 'stru' list.
You can now use the 'interprete' command to dump the contents of
a task structure (for example) :

< l task <enter>
> Task node name      : Node     Pri StackPtr  StackS Stat Command        Acc
> ------------------------------------------------------------------------
> Background Process  : 07E28330 00  07E2D500    4096 Wait iprefs     (02) -
```

```
> RexxMaster          : 07E51438 04  07E51C7A    2048 Wait            (00) -
> ...
> RAM                 : 07E23BF8 0A  07E23EE6    1200 Wait           PROC -
> input.device        : 07E08B22 14  07E09B28    4096 Wait           TASK -
> Background Process  : 07E1F268 04  07E8B94A   12000 Run  pv        (01) -

< interprete 07E1F268 tc
> FLAGS        : 00       | STATE        : 02       | IDNESTCNT    : FF
> TDNESTCNT    : FE       | SIGALLOC     : FF80FFFF | SIGWAIT      : 00000020
> SIGRECVD     : 20800100 | SIGEXCEPT    : 00000000 | ETask        : 80000000
> EXCEPTDATA   : 00000000 | EXCEPTCODE   : 00F83AEC | TRAPDATA     : 00000000
> TRAPCODE     : 07E7BE2C | SPREG        : 07E8B904 | SPLOWER      : 07E88DE8
> SPUPPER      : 07E8BCC8 | MEMENTRY     : 07E1F2AA | Userdata     : 000007E0
```

Related commands:
addstruct

remstruct

view
Related functions: peek() apeek() stsize()

Related lists: stru

Related tutor chapters: Looking at things

1.60 Command Reference : kill

```
Kill <task>|<crash node>
```

Cancel the specified task. This command works even if the task was frozen.

If the task you want to kill has crashed, PowerVisor will also remove the corresponding crash node. You can also kill a crashed task with a pointer to the crash node instead of the task node.
Do not kill a debug task. Remove this with the
debug
command.

Note that 'kill' will also try to remove all windows and screens belonging to the task.

Note that it is very dangerous to kill a task while it is using some functions from the dos.library.

'kill' uses autodefault to the 'task' list for the first argument.

Related commands:
debug

freeze

```
                Related lists:  task   crsh
```

1.61 Command Reference : led

```
                   LEd


Use this command to toggle the powerled on or off. This is useful when
a program is crashed or when you are using the led monitor.


Related commands:
                 addfunc
                Related lists:  func
```

1.62 Command Reference : libfunc

```
                   LIBFunc <library> <offset>


This command shows you the name of the library function corresponding
with a library and an offset (offset must be negative).
You must have loaded the corresponding fd-file first.
Note that only the 16 least significant bits of <offset> are used so
it is safe to use $ffe2 for example.

This command looks a bit like the reverse of
                 libinfo
                .

'libfunc' uses autodefault to the 'libs' list for the <library> argument.

This command returns a string if called from ARexx.


Related commands:
                 libinfo

                 loadfd

                 unloadfd
                Related lists:  fdfi   libs
```

1.63 Command Reference : libinfo

```
                   LIBInfo <library function name>


This command shows you the library, the offset and the register usage
for a library function. You must have loaded the corresponding fd-file
```

```
first.

< loadfd exec fd:exec_lib.fd <enter>
> ...

< libinfo putmsg <enter>
> 07E007CC   -366 (A0,A1)


Related commands:
                 loadfd

                 unloadfd

                 libfunc
                Related lists:  fdfi

Related tutor chapters:  Looking at things
```

1.64 Command Reference : list

```
                  List [<list>]


With arguments this command lists the current list. With an argument
'list' lists <list> :-)


Related commands:
                 info
                Related tutor chapters:  List Reference   Getting Started
```

1.65 Command Reference : llist

```
                  LList <list> ['start']


Traverse a system list and print all nodes in the list.
If 'start' is specified, 'LList' will start listing from the start of the
list.

You can type really anything for the 'start' argument. The simple presence
of an extra argument is enough.


Example :

< list port <enter>
> MsgPort node name   : Node     Pri SigBit SigTask
> ------------------------------------------------------------------------
> REXX                : 07E444A4 00      31 07E51438
> AREXX               : 07E51CC8 00      30 07E51438
```

```
> AddTools by Steve Ti: 07E42A50 00      31 07E70C98
> PowerVisor-port     : 07E7C6F6 00       1 00000000
> REXX_POWERVISOR     : 07E22098 00      24 07E1F268
> * Blank_Port        : 07E43390 00      30 07E73E28
> IPrefs.rendezvous   : 07E227F0 E2      31 07E28330
> SetPatch Port       : 07E227C0 9C       0 00000000

< llist port:rexx_powervisor <enter>
> Node name           : Node     Pri
> ------------------------------------------------------------------------
> * Blank_Port        : 07E43390 00
> IPrefs.rendezvous   : 07E227F0 E2
> SetPatch Port       : 07E227C0 9C

< llist port:rexx_powervisor start <enter>
> Node name           : Node     Pri
> ------------------------------------------------------------------------
> REXX                : 07E444A4 00
> AREXX               : 07E51CC8 00
> AddTools by Steve Ti: 07E42A50 00
> PowerVisor-port     : 07E7C6F6 00
> REXX_POWERVISOR     : 07E22098 00
> * Blank_Port        : 07E43390 00
> IPrefs.rendezvous   : 07E227F0 E2
> SetPatch Port       : 07E227C0 9C


Related commands:
                 list
               Related tutor chapters:  Looking at things
```

1.66 Command Reference : load

```
                  <bytes loaded> <- LOAd <filename> <start> [<max bytes>]


Load a file into memory. Only <max bytes> are read in if specified.
Please allocate the memory you use.

This command displays the actual number of bytes read.


Example :

< pointer=alloc(n,10000) <enter>

< load s:startup-sequence pointer 10000 <enter>
> 0000066C , 1644

< disp pointer <enter>
> 07EA7E4A , 132808266

You can free your memory later with the  free()  function. PowerVisor will
automatically free this memory when you quit or use the
                 cleanup
```

```
                command.


Related commands:
                save

                cleanup
               Related functions:  free()   alloc()

Related tutor chapters:  Scripts
```

1.67 Command Reference : loadfd

```
                  RC,<number of functions> <- LOADFd <library> <file-name>


This command loads a fd-file in memory. After you have loaded an fd-file
you can use the library functions defined in it.

PowerVisor uses fd-files for the following features :

   - You can call library functions defined in the fd-file. These library
     functions can be called as you would call them using C
   - PowerVisor will know how to display library functions when you are
     debugging. This is very useful
   - You can use all loaded functions with the
                addfunc
                command
   - You can ask information about all loaded library functions with
     the
                libfunc
                and
                libinfo
                commands

'loadfd' will do nothing when the fd-file is already loaded in memory.
You can find all loaded fd-files in the 'FdFi' list.

'rc' contains the pointer to the loaded fd-file node in the 'FdFi' list.
<number of functions> is equal to the number of functions actually loaded.
'rc' will be equal to -1 when the fd-file already existed.


Example :

< loadfd exec fd:exec_lib.fd <enter>
> New functions: 0000007E,126

< libinfo allocmem <enter>
> 07E007D8   -198 (D0,D1)


Related commands:
                unloadfd
```

```
                libinfo

                libfunc

                addfunc
              Related lists:  fdfi
```

1.68 Command Reference : loadtags

```
                  LOADTags <file> <base>
```

Load tags from a file to the current tag list. All tags in the file are added to the tags in the current tag list. When an old tag collides with a new tag, the old tag will be removed (two tags collide when they use the same address).

If the tagfile contains structures (type 'ST') you must load these structure in memory (with

```
                addstruct
```

) before loading the tags. Otherwise the type of the tag will be changed to Long/Ascii ('LA') and you will get a warning :

```
   'Warning ! Unknown structure types have been changed to LA !'
```

<base> is a pointer that you can use to relocate tags. A tag file also contains a base. When the base in the tag file and <base> are equal the tags will be on the same position as when they were saved. <base> is useful when you want to pause your work and have to reloade them at another time. Since you can't always make sure that things are on the same position you will have to load the tags relative to some base.

Related commands:

```
                savetags

                addtag

                remtag

                cleartags

                view

                tg

                usetag

                checktag

                tags
```

```
                addstruct
               Related functions:  taglist()

Related tutor chapters:  Looking at things
```

1.69 Command Reference : locate

```
                  LOCAte <x> [<y>]


Adjust the current location on the current logical window.


Example :

< locate 0,0 <enter>
< disp 3 <enter>
> 00000003 , 3


Related commands:
                home

                cls

                print

                current
               Related functions:  getx()   gety()   getchar()   lines()   cols()
                   getlwin()

Related lists:  lwin

Related tutor chapters:  Screens and windows
```

1.70 Command Reference : log

```
                  LOG [<logical window> <filename>]


This command enables or disables the logging of all output in a logical
window to a file. PowerVisor only supports one log file. If you open one
for a logical window, PV will automatically close the other log file (if
there is one).

'log' uses autodefault to the 'lwin' list for the first argument.

You can disable (and close) command logging if you give no arguments to
this command.

You can also use the
                to
```

```
                command to make a temporary log for one command
(or a group of commands).

When you use the '-' prefix in front of the commandline, PowerVisor will
show no output on the current logical window. The output will only go to
the file. When you use the '~' prefix in front of the commandline,
PowerVisor will not show the feedback line on the screen (and also not in
the file).


Related commands:
                to
               Related lists:  lwin

Related tutor chapters:  Screens and windows
```

1.71 Command Reference : memory

```
                  Memory [<start> [<bytes>]]


This command shows memory beginning at <start>.
<bytes> bytes are shown.

With no arguments this command continues the memory list.

<bytes> is 320 by default.
PowerVisor remembers the last number of bytes used with the command
(and the
                view
                command) and uses this number as the new default
number of <bytes> for the following 'memory' (or 'view') commands.

Use the
                mode
                command to install preferences for list memory:
   mode byte         for byte grouping
   mode word         for word grouping
   mode long         for longword grouping (default)
   mode ascii        for ascii viewing

Look at the
                view
                command for a more powerful dump routine.

If you press <enter> after a 'memory' command, the listing will
continue.


Related commands:
                view

                unasm

                mode
```

```
                Related functions:  lastmem()   lastbytes()

Related tutor chapters:  Looking at things
```

1.72 Command Reference : memtask

```
   MEMTask <task>


Show memory for a task. This is the memory which is allocated via
AllocEntry and is attached to the task.

'memtask' uses autodefault for the 'task' list for the first argument.


Example :

< memtask powervisor <enter>
> 07EB4F28,132861736
>   07EB4F38         92
>   07EB4F40       4096

This means that there is one memory header node in the TC_MEMENTRY
list of the task. This node contains two entries. One with 92 allocated
bytes and one with 4096 allocated bytes. The first number in the
output is the address of the allocated memory.


Related lists:  task

Related tutor chapters:  Looking at things
```

1.73 Command Reference : mmuregs

```
                   MMURegs


(only 68020 with 68851, 68030 or 68040)
Show all special mmu registers (except MMUSR).

If some register is not available on your Amiga, '(na)' is printed
after the register name (for example, 'DRP' is only available on the
68851)


Example :

< mmuregs <enter>
> DRP  : (na)
> CRP  : 000F0002   07FFF140
>    L/U bit is cleared
>    LIMIT = 0000000F
```

```
>    DT    = Valid 4 byte
>    Table address = 07FFF140
> SRP  : 80000001   00000000
>    L/U bit is set
>    LIMIT = 00000000
>    DT    = Page descriptor
>    Table address = 00000000
> TC   : 80F08630
>    Enable address translation
>    Disable Supervisor Root Pointer (SRP)
>    Disable Function Code Lookup (FCL)
>    System page size    = FFFF8000
>    Initial shift       = 00000000
>    Table Index A (TIA) = 00000008
>    Table Index B (TIB) = 00000006
>    Table Index C (TIC) = 00000003
>    Table Index D (TID) = 00000000
> TT0  : 04038207
>    Log Address Base = 00000004
>    Log Address Mask = 00000003
>    TT register enabled
>    No Cache Inhibit
>    R/W set
>    RWM cleared
>    FC value for TT block = 00000000
>    FC bits to be ignored = 00000007
> TT1  : 403F8107
>    Log Address Base = 00000040
>    Log Address Mask = 0000003F
>    TT register enabled
>    No Cache Inhibit
>    R/W cleared
>    RWM set
>    FC value for TT block = 00000000
>    FC bits to be ignored = 00000007


Related commands:
                 mmutree

                 mmurtest

                 mmuwtest

                 specregs
                Related tutor chapters:  Looking at things
```

1.74 Command Reference : mmureset

```
                   MMURESet


(only 68020 with 68851, 68030 or 68040)
This command resets all U and M flags in the MMU tree to false.
You can then use the
```

```
                mmutree
                command to examine which pages are
modified and used.

(Note this command does not support all possible mmu tables, therefore
it does not work in AmigaDOS 1.3)


Related commands:
                mmutree

                mmuregs
               Related tutor chapters:  Looking at things
```

1.75 Command Reference : mmurtest

```
                  MMURTest <address>


(only 68020 with 68851, 68030 or 68040)
Test an address for read access in MMU tree.
MMUSR is dumped. (does not work yet)


Related commands:
                mmutree

                mmuregs

                mmuwtest

                specregs
               Related tutor chapters:  Looking at things
```

1.76 Command Reference : mmutree

```
                  MMutree


(only 68020 with 68851, 68030 or 68040)
This command shows the current MMU tree (CRP only).

FC code trees are not implemented yet.
8 byte format pages are not implemented yet.
indirect pages are not implemented yet.


Example :

< mmutree <enter>
> 00000000   4 BYTE (imuw)  Log: 00000000 # 00000000
> 07FFF140      4 BYTE (imUw)  Log: 00000000 # 01000000
```

```
> 07FFF180         PAGE   (IMUw)  Log: 00000000 # 00040000  -> 00000000
> 07FFF184         PAGE   (IMUw)  Log: 00040000 # 00040000  -> 00040000
> 07FFF188         PAGE   (IMUw)  Log: 00080000 # 00040000  -> 00080000
> ...
> 07FFF268         PAGE   (IMUw)  Log: 00E80000 # 00040000  -> 00E80000
> 07FFF26C         PAGE   (IMUw)  Log: 00EC0000 # 00040000  -> 00EC0000
> 07FFF270         PAGE   (iMUw)  Log: 00F00000 # 00040000  -> 00F00000
> 07FFF274         PAGE   (iMUw)  Log: 00F40000 # 00040000  -> 00F40000
> 07FFF278         PAGE   (iMUW)  Log: 00F80000 # 00040000  -> 07F80000
> 07FFF27C         PAGE   (iMUW)  Log: 00FC0000 # 00040000  -> 07FC0000
> 07FFF144      PAGE   (iMUw)  Log: 01000000 # 01000000  -> 01000000
> 07FFF148      PAGE   (iMUw)  Log: 02000000 # 01000000  -> 02000000
> 07FFF14C      PAGE   (iMUw)  Log: 03000000 # 01000000  -> 03000000
> 07FFF150      PAGE   (iMUw)  Log: 04000000 # 01000000  -> 04000000
> 07FFF154      PAGE   (iMUw)  Log: 05000000 # 01000000  -> 05000000
> 07FFF158      PAGE   (iMUw)  Log: 06000000 # 01000000  -> 06000000
> 07FFF15C      4 BYTE (imUw)  Log: 07000000 # 01000000
> 07FFF280         INV    (imuw)  Log: 07000000 # 00040000
> 07FFF284         INV    (imuw)  Log: 07040000 # 00040000
> 07FFF288         INV    (imuw)  Log: 07080000 # 00040000
> ...
> 07FFF344         INV    (imuw)  Log: 07C40000 # 00040000
> 07FFF348         INV    (imuw)  Log: 07C80000 # 00040000
> 07FFF34C         INV    (imuw)  Log: 07CC0000 # 00040000
> 07FFF350         PAGE   (iMUw)  Log: 07D00000 # 00040000  -> 07D00000
> 07FFF354         INV    (imuw)  Log: 07D40000 # 00040000
> 07FFF358         INV    (imuw)  Log: 07D80000 # 00040000
> 07FFF35C         INV    (imuw)  Log: 07DC0000 # 00040000
> 07FFF360         PAGE   (iMUw)  Log: 07E00000 # 00040000  -> 07E00000
> 07FFF364         PAGE   (iMUw)  Log: 07E40000 # 00040000  -> 07E40000
> 07FFF368         PAGE   (iMUw)  Log: 07E80000 # 00040000  -> 07E80000
> 07FFF36C         PAGE   (iMUw)  Log: 07EC0000 # 00040000  -> 07EC0000
> 07FFF370         PAGE   (iMUw)  Log: 07F00000 # 00040000  -> 07F00000
> 07FFF374         PAGE   (iMUw)  Log: 07F40000 # 00040000  -> 07F40000
> 07FFF378         PAGE   (iMUW)  Log: 07F80000 # 00040000  -> 07F80000
> 07FFF37C         PAGE   (iMUW)  Log: 07FC0000 # 00040000  -> 07FC0000
> 07FFF160      PAGE   (iMUw)  Log: 08000000 # 01000000  -> 08000000
> 07FFF164      PAGE   (iMUw)  Log: 09000000 # 01000000  -> 09000000
> 07FFF168      PAGE   (iMUw)  Log: 0A000000 # 01000000  -> 0A000000
> 07FFF16C      PAGE   (iMUw)  Log: 0B000000 # 01000000  -> 0B000000
> 07FFF170      PAGE   (iMUw)  Log: 0C000000 # 01000000  -> 0C000000
> 07FFF174      PAGE   (iMUw)  Log: 0D000000 # 01000000  -> 0D000000
> 07FFF178      PAGE   (iMUw)  Log: 0E000000 # 01000000  -> 0E000000

Some entries explained :

> 07FFF140      4 BYTE (imUw)  Log: 00000000 # 01000000

   Address for this mmu entry is 07FFF140
   It describes a 4 BYTE page descriptor
   The descriptors are not invalid (i)
   The descriptors are not modified (m)
   The descriptors are used (U)
   The descriptors are not write protected (w)
   Logical address is 00000000
   The page describes 01000000 bytes of memory
```

```
> 07FFF27C         PAGE   (iMUW)  Log: 00FC0000 # 00040000  -> 07FC0000

   Address for this mmu entry is 07FFF27C
   It describes a PAGE
   The page is not invalid (i)
   The page is modified (M)
   The page is used (U)
   The page is write protected (W)
   Logical address is 00FC0000 (this is in pseudo ROM on an Amiga 3000)
   The page describes 00040000 bytes of memory
   The page is mapped to the 07FC0000 physical address (fast RAM)


Related commands:
                 mmuregs

                 mmurtest

                 mmuwtest

                 specregs

                 mmureset
                Related tutor chapters:  Looking at things
```

1.77 Command Reference : mmuwtest

```
                   MMUWtest <address>


(only 68020 with 68851, 68030 or 68040)
Test an address for write access in MMU tree.
MMUSR is dumped. (does not work yet)


Related commands:
                 mmutree

                 mmuregs

                 mmurtest

                 specregs
                Related tutor chapters:  Looking at things
```

1.78 Command Reference : mode

```
                   MOde <mode argument> {<mode argument>...}


Install preferences for PowerVisor.
```

```
The following arguments are supported:
   pal          set monitor to pal (only AmigaDOS 2.0)
                the non-interlaced resolution is 640x256 and the
                interlaced resolution is 640x512. This resolution will be
                bigger if you use overscan.
   ntsc         set monitor to ntsc (only AmigaDOS 2.0)
                non-interlaced : 640x200
                interlaced : 640x400
   vga          set monitor to vga (only AmigaDOS 2.0 and new denise)
                non-interlaced : 640x480
                interlaced : 640x960
   viking       set monitor to a2024 (only AmigaDOS 2.0)
                resolution : 1024x1008

   lace         use interlace
   nolace       no interlace (default)

   fancy        use two bitplanes for the PowerVisor screen (default)
   nofancy      use only one bitplane

   sbottom      include the size gadget in the bottom border. This means
                that you loose a line but you gain some columns (default)
   nosbottom    include the size gadget in the right border of the window.
                You loose some columns but you gain a line or so

   space        add a space after a snapped word (default)
   nospace      don't add a space. Simply snap the word as it is

   lonespc      snap a space if you click on an empty place in a logical
                window
   nolonespc    don't snap a space (default)

   shex         show hex words when disassembling instructions. The
                disadvantage is that these words could overwrite the right
                side of the symbolname if present (default)
   noshex       don't show hex words. Symbols will be completely visible

   dec          display all printed integers as decimal only
   hex          display all printed integers as hexadecimal only
   hexdec       first display hex, than display decimal (default)

   more         enable -MORE- check for the 'Main' logical window (default)
   nomore       disable -MORE- check

   auto         perform an automatic list whenever the current list changes
   noauto       don't do this (default)

   byte         list memory as bytes (for the
                  memory
                  command)
   word         list memory as words
   long         list memory as longs (default)
   ascii        list memory as ascii

   fb           feedback each command as it is typed in by the user on
                the current logical window (default)
   nofb         don't do this
```

```
   patch       patch the Exec AddTask function. When this function is
               patched by PowerVisor crashtrapping will work better
               for all new tasks created after the patch is applied.
               This is recommended if you use resident breakpoints
               (see the  Debugging  chapter). Note that it is not safe to
               run other debuggers (like CodeProbe or MonAm) when
               the patch is applied. They will probably crash when
               you try to trace with them. There will (probably) be
               no problems if you start the other debugger and load
               the debug program with this debugger BEFORE you
               apply the patch (before you start PowerVisor or before
               you type 'mode patch'). (default)
               See also the
                 crash
                 command
   nopatch     Don't patch the Exec AddTask function. When the AddTask
               function is not patched, PowerVisor will trap a crash
               a bit later (too late if you plan to use resident
               breakpoints). With the patch applied PowerVisor traps
               crashes just on the spot while if the patch is not
               applied PowerVisor will only trap the crash just before
               the guru would normally arrive (you will even have to
               press 'cancel' on the 'task-held' requester before
               PowerVisor notices the crash). But 'mode nopatch' is
               the only safe way to run other debuggers concurrently
               with PowerVisor.

   intui       if this option is set, PowerVisor will also open a
               physical window (or Intuition window) everytime you
               open one of the standard logical windows ('Extra',
               'Debug', ...) with the standard commands (
                 xwin
                ,

                 dwin
                , ...) (not with
                 openlw
                ). This is useful if
               you prefer to work with Intuition windows instead of
               PowerVisor logical windows. The logical window
               is of course opened in this physical window (with the
               same name).
   nointui     Simply open each standard logical window in the 'Main'
               physical window (default)


You can set and examine the 'mode' variable. This is useful for scripts
(for example to remember the current settings so that the script can
restore them later).

When you have installed everything as you like it best you can make
the new values default with the
                 saveconfig
                 command.
```

```
Example :

< mode byte dec <enter>

< d 4 <enter>
> 4

< m 0 <enter>
> 00000000: 00 00 00 00 07 E0 07 CC 00 F8 08 34 00 F8 0B 16  ...........4....
> 00000010: 00 F8 0A DA 00 F8 0A DC 00 F8 0A DE 00 F8 0A E0  ................
> 00000020: 00 F8 0C 00 00 F8 0A E4 00 F8 0A E7 00 F8 0A E8  ................
> 00000030: 00 F8 0A EA 00 F8 0A EC 00 F8 0A EE 00 F8 0A F0  ................
> 00000040: 00 F8 0A F2 00 F8 0A F4 00 F8 0A F6 00 F8 0A F8  ................
> ...

< mode long hexdec <enter>

< d 4 <enter>
> 00000004,4

< m 0 <enter>
> 00000000: 00000000 07E007CC 00F80834 00F80B16              ...........4....
> 00000010: 00F80ADA 00F80ADC 00F80ADE 00F80AE0              ................
> 00000020: 00F80C00 00F80AE4 00F80AE7 00F80AE8              ................
> 00000030: 00F80AEA 00F80AEC 00F80AEE 00F80AF0              ................
> 00000040: 00F80AF2 00F80AF4 00F80AF6 00F80AF8              ................
> 00000050: 00F80AFA 00F80AFC 00F80AFE 00F80B00              ................
> 00000060: 00F80B02 00F810F4 00F81152 00F81188              ...........R....
> 00000070: 00F811E6 00F8127C 00F812C6 00F81310              .......|........
> 00000080: 00F80B70 00F80B72 00F80B74 00F80B76              ...p...r...t...v
> 00000090: 00F80B78 00F80B7A 00F80B7C 00F80B7E              ...x...z...|...~
> 000000A0: 00F80B80 00F80B82 00F80B84 00F80B86              ................
> 000000B0: 00F80B88 00F80B8A 00F80B8C 00F80B8E              ................
> 000000C0: 00F80B90 00F80B92 00F80B94 00F80B96              ................
> 000000D0: 00F80B98 00F80B9A 00F80B9C 00F80B9E              ................
> 000000E0: 00F80BA0 00F80BA2 00F80BA4 00F80BA6              ................
> 000000F0: 00F80BA8 00F80BAA 00F80BAC 00F80BAE              ................
> 00000100: 80000005 07E08B22 66FFE6F7 00000000              ........"f......
> 00000110: 66FF66FA 66FB667F 66FFA67F 66FF66F1              f.f.f.ff..f.f.
> 00000120: 66FFF6FB 66F7E6DF 66FF663F 66FF66F7              f...f...f.f?f.f.
> 00000130: 66FF66F9 64FF66FF 66FF66FF 66DFA6E7              f.f.d.f.f.f.f...


Related commands:
                 saveconfig

                 prefs
                Related tutor chapters:  Installing PowerVisor
```

1.79 Command Reference : move

```
                   MOVe <physical window> <x> <y>
```

Move the physical window to a specified position. This command only works

```
if the physical window is a non-backdrop window.
You can move PowerVisor to another screen with the
                 screen
                 command.
You can of course also move the PowerVisor window using the drag gadget.

'move' uses autodefault to the 'pwin' list for the first argument.


Related commands:
                 screen

                 size
               Related lists:  pwin

Related tutor chapters:  Screens and windows
```

1.80 Command Reference : next

```
                  RC,<address> <- Next


Continue searching at the last position.
This command prints 0 if not found.

You can use the  lastfound()  function to see where 'next' will continue
its search.


Related commands:
                 search

                 copy

                 fill
               Related functions:  lastfound()
```

1.81 Command Reference : on

```
                  ON <logical window> <command>


With this command you can execute an other command, so that it's output
appears on the logical window of your choice (except 'Debug' and 'Source').
This command temporary sets the current window to <logical window>.

'on' uses autodefault to the 'lwin' list for the first argument.

   on <log win> <command>

      is the same as
```

```
   current <log win>
   <command>
   current <previous log win>


Example :

open 'Extra' logical window :

< xwin <enter>

execute a list on that window :

< on extra list task <enter>
> ...

Output appears on 'Extra'.

Using the group operator you can execute more commands at once :

< on extra {disp 1;disp 2;disp 3} <enter>
> 00000001 , 1
> 00000002 , 2
> 00000003 , 3

appears on 'Extra'.

You can also execute a command in a string :

Wait for input from user :

< scan <enter>

Type command (at the input prompt = '????>') :

< disp 4 <enter>

< on extra #input <enter>
> 00000004 , 4


Related commands:
                 xwin

                 current

                 openlw

                 closelw
                Related functions:  getlwin()

Related lists:  lwin

Related tutor chapters:  Screens and windows
```

1.82 Command Reference : opendev

```
                  RC,<pvdevice> <- Opendev <device name> [<unit> [<flags>]]


Open a device for use with the
                 closedev
                ,
                 devinfo
                 and
                 devcmd
                 commands.
The result (in RC) is a pointer to a block containing the pointer to the
port and the pointer to the IORequest (256 bytes big) in that order.
We call this block a PVDevice.
Use this block as the argument to
                 closedev
                ,
                 devinfo
                 and
                 devcmd
                .


Example :

Open the device :

< dev={opendev "trackdisk.device"} <enter>

Put the drive motor on :

< devcmd dev 9 0 1 <enter>

Put the drive motor off :

< devcmd dev 9 0 0 <enter>

Ask more information :

< devinfo dev <enter>
> MsgPort node name   : Node     Pri SigBit SigTask
>                     : 07E9D2A0 00      20 07EB5948
>
> IORequest    : 07E9E130 | Succ         : 00000000 | Pred         : 00000000
> Type         : 05       | Pri          : 00       |
> Name         :
> ReplyPort    : 07E9D2A0 | MN_Length    : 0100     | Device       : 07E077AC
> Unit         : 07E0C5A0 | Command      : 0000     | Flags        : 00
> Error        : 00       | Actual       : 00000000 | Length       : 00000000
> Data         : 00000000 | Offset       : 00000000 |

Close the device :

< closedev dev <enter>
```

```
Related commands:
                closedev

                devcmd

                devinfo
```

1.83 Command Reference : openlw

```
                  RC,<lwin> <- OPENLw <physical window> <logwin name> <cols> < ↩
                     rows>
               [<brother> <where> [<number of columns or lines>]]


Open a logical window on a physical window. If the physical window is empty
you need not specify the two last arguments. Otherwise <brother> is the
logical window which will be your brother and <where> specifies where
our new logical window should appear relative to the <brother>.
The <where> string can be like :

   u     above brother
   d     below brother
   r     right from brother
   l     left from brother
   pd    take parent from brother and appear below this box
   pppr  take parent from parent from parent and appear right from this box
   ...

'openlw' uses autodefault to the 'pwin' list for the <physical window> arg.
'openlw' uses autodefault to the 'lwin' list for the <brother> argument.

Note that you can open standard physical windows with this command as well,
instead of using the predefined commands (xwin,rwin,awin,owin,dwin,swin).

If <number of columns or lines> is specified it is used as the number
of lines or columns for the new logical window (<number of columns or
lines> is interpreted as <number of columns> if the new window is
positioned left or right of <brother>, otherwise <number of columns or
lines> is interpreted as <number of lines>).

'rc' and <lwin> are the pointer to the new logical window.


Example :

Open the 'Extra' logical window at the left of the 'Main' logical window :

< openlw main Extra 80 40 main l <enter>

And to close the logical window :

< closelw extra <enter>
or
< xwin <enter>
```

```
Related commands:
                 closelw

                 awin

                 rwin

                 owin

                 dwin

                 swin

                 xwin

                 openpw

                 closepw
                Related lists:  lwin   pwin

Related tutor chapters:  Screens and windows
```

1.84 Command Reference : openpw

```
                   RC,<pwin> <- OPENPw <physical window name> <x> <y> <w> <y>


Open a physical window. It will be empty (no logical windows).
You can move and size this window using
                 move
                 and
                 size
                . When
you use the
                 screen
                 command to move PowerVisor to another screen,
all physical windows will automatically move together with the
'Main' physical window.

You can only open 5 additional physical windows since there are only
5 signals left. In a later version this limitation could be removed.

'rc' and <pwin> are the pointer to the new physical window.


Related commands:
                 closepw

                 openlw

                 closelw
                Related lists:  pwin
```

Related tutor chapters: Screens and windows

1.85 Command Reference : owin

OWin [<number of lines>]

Open/close the PortPrint window. When the 'PPrint' logical window is open,
PowerVisor will use it for all PortPrint output. Otherwise the current
logical window is the one that will receive the output.

Normally the height of the new logical window is 30 % of the total
physical window height ('Main' is the physical window for all standard
logical windows). However, if you specify <number of lines>, the logical
window will be opened with <number of lines> visible lines (This is
the number of lines when the default PowerVisor font is used).

By default the 'PPrint' logical window has the following characteristics :

- Number of columns is fixed and equal to the maximum number of columns
  visible at the time the logical window is created
- Number of rows is fixed and is always equal to 50
- -MORE- checking is disabled
- Interrupt/Pause checking is disabled
- Home position is top-visible
- Auto Output Snap is off

You can change these characteristics with the
prefs
command.

Note that if the 'intui' mode flag is one (this is off by default, see the

mode
command) the logical window will be opened on a new physical ↩
window.
<number of lines> is ignored in that case.

Related commands:
rwin

dwin

swin

awin

xwin

fit

colrow

prefs

```
                 mode
               Related lists:  lwin

Related tutor chapters:  Screens and windows
```

1.86 Command Reference : owner

```
   OWNer <address>


This command tries to see who is the owner of the address you specify.
This command is not completely ready but it is safe to use.

At this moment the 'owner' command searches the task list to see if it
belongs to a task or process. The stack, seglists, task structure,
process structure and cli structure are checked. In future more
checks will be made.


Example :

< l task <enter>
> Task node name      : Node     Pri StackPtr  StackS Stat Command        Acc
> ------------------------------------------------------------------------
> Background Process  : 07E4F080 00  07E6D640    4096 Rdy  clock      (04) -
> Background Process  : 07E28330 00  07E2D500    4096 Wait iprefs     (02) -
> RexxMaster          : 07E51438 04  07E51C7A    2048 Wait            (00) -
> SYS:System/CLI      : 07E45DC0 00  07E46CFE    4096 Wait            (00) -
> ...
> PowerSnap 1.0 by Nic: 07E629C0 05  07E6320A    2000 Wait           PROC -
> Background Process  : 07E70C98 00  07E6E8EA    4096 Wait addtools   (07) -
> input.device        : 07E08B22 14  07E09B28    4096 Wait           TASK -
> RAM                 : 07E23BF8 0A  07E23EE6    1200 Wait           PROC -
> Background Process  : 07E1F268 04  07E8BB76   12000 Run  pv         (01) -

< owner task:rexxmaster+6
> Found in TCB
> RexxMaster          : 07E51438 04  07E51C7A    2048 Wait            (00) -

(TCB is task control block)


Related tutor chapters:  Looking at things
```

1.87 Command Reference : pathname

```
                   PAthname <lock>


This command prints the pathname for a lock. If <lock> does not point to
a lock, 'pathname' prints an error. Note that <lock> must be an APTR and
```

```
not a BPTR !

This command returns a string if used from ARexx.


Related commands:
                unlock
               Related lists:  lock   fils
```

1.88 Command Reference : prefs

```
                  PREfs ('history' | 'key' | 'screen' | 'stack' | 'logwin' | ' ↩
                     linelen' |
         'debug' | 'dmode' | 'pens' | 'font') [<arguments> ...]


You can set/get preferences with this command. All things you install with
this command can be saved with the
                saveconfig
                command.
If you don't specify the optional arguments in the following templates, you
will get the current values.

Warning ! Changing preferences values is not always safe. For some
preferences (especially 'font' and 'screen') it is best to
set the preferences,
                saveconfig
                them,
                quit
                PowerVisor and restart
PowerVisor. Only after a full restart will PowerVisor correctly account
for all changes. 'history', 'key', 'stack', 'logwin', 'debug',
'dmode' and 'pens' are save. You may change them as much as you like,
PowerVisor will not get confused.
Changing 'linelen' has no effect at all. You have to quit to see
the changes.

The following preferences arguments are allowed :

   - history [<history value>]
         Install the maximum number of lines in the history. Default is 20

   - key <key number> [<code> <qualifier>]
         nr name                 default key           code    qualifier
         ------------------------------------------------------------------
         0  interrupt key        <esc>                 045     0
         1  hot key              <r-shift>+<r-alt>+'/' 03A     022
         2  pause key            <r-alt>+<help>        05F     020
         3  cycle active logwin  <tab>                 042     0
         4  history up           <arrow up>            04C     0
         5  history down         <arrow down>          04D     0

   - screen [<w> <h>]
         Define the width and the height of the PowerVisor screen. If you
         use -1 for any of these parameters the default screen width or
```

```
      height will be used.
      If you have AmigaDOS 2.0 you can install a big screen. You can
      scroll in this screen by moving the mousepointer out of the
      visible area. The default width and height are :
         -1,-1

- stack [<stack fail level>]
      Set the amount of stackspace left before PowerVisor will halt
      the task and give a warning (only if
              account
              is on)
      default value is 40
      This value is also used by the
              stack
              command.

- logwin <full standard logwin name> [<cols> <rows> <mask> <flags>]
      supported logwins :
         Main,Extra,Refresh,Debug,Rexx,PPrint
      bits in flag :
         4   = -MORE- enabled
         32  = real-top
         64  = statusline off
         128 = interrupt off
         256 = auto output snap

- linelen [<line length>]
      Set the maximum length of the stringgadget commandline. This is
      also the maximum number of characters that may be used in scripts
      (not ARexx scripts).
      default value is 400

- debug [<instructions> <show previous instruction>]
      <instructions> is the number of instructions to disassemble after
      each trace (default is 5)
      if <show previous instruction> is 1, the previous instruction
      is disassembled (default is 1)

- dmode 'n'|'r'|'c'|'f'
      Set the information that should be displayed after a trace. This
      parameter is not used by the fullscreen debugger since the
      fullscreen debugger always shows all information.
         n        show no info at all. This is useful when you are using
                  the fullscreen debugger and you do not want to be
                  disturbed by output on the current logical window
         r        show only registers
         c        show only code
         f        show registers and code (default)

- pens [<pen number> <pen color>]
      With this command you can install all colors used in PowerVisor.
      There are 48 pens. The first 24 are for fancy screens
      (2 bitplanes) and the last 24 are for 1 bitplane screens. Only
      the first 21 pens are used for both fancy and no-fancy screens.
      All other pens are reserved for future use.
      With no arguments this command lists all pens on two lines: the
      first line containing all pens for fancy screens and the second
```

line for no-fancy screens.

Here follow the currently defined pen numbers and there default color values. Add 24 to the pen number to get the no-fancy screen pen number (see the Installing PowerVisor chapter for the meaning of each pen).

nr	name	default fancy	default no-fancy
0	BoxBackground	0	0
1	LogWinBackground	0	0
2	NormalText	1	1
3	PrompText	1	1
4	StatusTextInActive	1	1
5	StatusTextActive	2	0
6	InActiveBar	0	0
7	ActiveBar	3	1
8	TopLeft3D	2	1
9	BottomRight3D	1	1
10	BoxLine	1	1
11	PositionBox	0	0
12	TopLeftBox	1	1
13	BottomRightBox	2	1
14	PositionIndicator	3	1
15	SGInActiveText	3	1
16	SGInActiveBackground	0	0
17	SGActiveText	1	1
18	SGActiveBackground	0	0
19	HilightPen	2	0
20	HilightBackPen	0	1

- font [<font name> <size> <style> <flags>]
  With this command you can install the default font used by PowerVisor. This is topaz.font,8,0,0 by default. <font name> must always end with '.font'. This font is used for all logical windows, for the stringgadget, for the screen titlebar and for the logical window titlebars.

See the Installing PowerVisor chapter for more information.

Related commands:
saveconfig

mode
Related tutor chapters: Installing PowerVisor Screens and ↩ windows

1.89 Command Reference : print

PRint <string>

Print a string on the current logical window. (Useful for scripts and

```
ARexx programs)


Example :

< print 'testing' <enter>
> testing

< print test <enter>
> test

To print a newline after the string, use:

< print 'testing\0a' <enter>

or

< print testing\0a <enter>

> testing

You can use the '\' for quoting characters too :

< print abc\ def\\"\'\ the\ end\0a <enter>
> abc def\"' the end

or you can use the \() operator :

< print 'number : \(1000+3)\0a' <enter>
> number : 1003

< print 'task : \('input.device',%08lx)\0a' <enter>
> task : 07E063A2


Related commands:
                 disp

                 locate

                 current

                 script

                 rx
                Related functions:  getlwin()

Related lists:  lwin

Related tutor chapters:  Getting Started
```

1.90 Command Reference : pvcall

```
   <result> <- PVcall <number> [<arguments>]
```

```
Call internal PowerVisor functions. This command is only useful for the
very experienced PowerVisor user. See the  The wizard corner  chapter
for more info.


Related tutor chapters:  The wizard corner   Scripts
```

1.91 Command Reference : quit

```
                  Quit


Quit PowerVisor. Everything will be cleaned up.
Frozen tasks are lost. Monitor functions will be cleaned. Crash trapping
is disabled. Stack checking is disabled. All debug tasks are frozen.
All FD-Files will be removed from memory.
If one of your debug tasks is tracing 'Quit' will print an error message.
First halt this task with 'trace h'.

Warning: Everything you allocated or opened using library functions will
         NOT be cleaned up. Remember to do it yourself.

         All crashed tasks are left in memory. They will simply wait.

         All freezed tasks are lost forever (and their memory too).
         UnFreeze or kill your tasks before you quit.

         All memory allocated with the  alloc()  function is
         automatically freed (This also includes memory allocated
         with
                 rblock
                )

         All tasks you were debugging will be frozen.


Related commands:
                 hold
                Related tutor chapters:  Getting Started
```

1.92 Command Reference : rblock

```
                  RC,<address> <- RBlock <Unit number> <block number> [<address>]


Read a block from a disk. This command will allocate the memory for it
if you do not specify <address>. The pointer to this memory will be
stored in the 'rc' variable. You can read a block in your own memory
if you specify the address. This address must be in chip ram.
You can free the memory this command allocated with  free() .
```

```
The memory allocated with this command is automatically freed when
PowerVisor quits. You can also free this memory with the
                  cleanup
                  command.


Related commands:
                  wblock

                  cleanup

                  showalloc
                 Related functions:  alloc()   free()
```

1.93 Command Reference : refresh

```
                    Refresh [<refresh rate> <command>]


This command installs <command> to be executed every <refresh rate>
IntuiTick (one IntuiTick is a tenth of a second).

With no arguments 'refresh' removes the refresh command.

Note that the output from <command> is send to the
'Refresh' logical window if it is open. Otherwise the output goes to
the current logical window. Use the
                  rwin
                  command to open the 'Refresh'
window.


Example :

< rwin <enter>
< a=0 <enter>
< refresh 1 {home;disp a;a=a+1} <enter>

Look and admire the result ...

< refresh <enter>

You can also refresh a command from a string :

Wait for input from user :

< scan <enter>

Type command (at the input prompt = '????>') :

????< disp 4 <enter>
< refresh 10 #input <enter>


Related commands:
```

```
                rwin

                list

                home

                openlw

                closelw
               Related functions:  rfrate()   rfcmd()

Related lists:  lwin

Related tutor chapters:  Getting Started   Screens and windows
```

1.94 Command Reference : regs

```
                  REGs <task>|<crash node>|<debug node>


Show the registers for a task, a crash node or a debug node.
When a task is crashed use the crash node pointer instead. If you don't
do this, some register may contain wrong information.


Example :

< regs task:input.device <enter>
> input.device        : 07E08B22 14  07E09B28    4096 Wait             TASK -
> ------------------------------------------------------------------------------
> D0: 07E0F1B4   D1: 00000000   D2: 40000000   D3: 00008000
> D4: 00000000   D5: 00000000   D6: 80000000   D7: C0000000
> A0: 07E08ACC   A1: 07E08B22   A2: 07E0D7A6   A3: 07E09CB6
> A4: 07E09B82   A5: 07E00796   A6: 07E007CC
> PC: 00F80C14   SP: 07E09B6E   SR: 0010


Related commands:
                fregs
               Related lists:  task   crsh   dbug
```

1.95 Command Reference : remattach

```
                  REMAttach <attach node>


Remove a macro.

'remattach' uses autodefault to the 'attc' list for the first argument.


Example :
```

```
Go to the attachement list :

< attc <enter>
< list <enter>

> Node     Code Qualifier Command
> ------------------------------------------------------------------------
> 00C184E0   81         2 'info task:'trackdisk.device' task               '
> 00C34268   80         0 'list                                            '

< remattach 00c34268 <enter>


Related commands:
                 attach

                 list
                Related lists:  attc

Related tutor chapters:  Installing PowerVisor   Scripts
```

1.96 Command Reference : remclip

```
                   REMCLip <Clip name>


Remove a clip previously set with
                 clip
                 or with some other external ARexx
program.


Related commands:
                 clip

                 rx

                 assign
                Related tutor chapters:  Scripts
```

1.97 Command Reference : remcrash

```
   REMCrash <crash node>


When a task crashes, PowerVisor makes a crash node. You can find this
node in the Crsh list. You can remove this node with the remcrash command.
Note that this command will not remove the task. The task will simply be
left waiting for a signal that will never come.
```

Related lists: crsh   task

1.98 Command Reference : remfunc

REMfunc <function monitor node>

Use this function to remove a patch you installed with 'addfunc'.
Note that you must remove multiple patches of the same function in
reverse order. Note that PowerVisor will give an error if you try
to remove the oldest patch first.

Warning ! Be very careful when it is possible that some other program
is also patching the function. There could be conflicts. PowerVisor
tries to be friendly and checks everything, but you can't be sure that
other programs are as friendly as PowerVisor.

You can find the nodes for the function monitor in the 'func' list

'remfunc' uses autodefault to the 'func' list for the first argument.

Related commands:
addfunc
Related lists: func

1.99 Command Reference : remhand

REMHand <input handler>

Remove an input handler. You can list the inputhandlers in the 'IHan' list.

'remhand' uses autodefault to the 'ihan' list for the first argument.

Related lists: ihan

1.100 Command Reference : remove

<node> <- REMOve <node>

Remove a node from a list. Be carefull with this command. No checking is
done if you are really removing a node. The memory for this node is not
freed (you can later include this node again in the list with the exec
functions 'addhead', 'addtail' or 'enqueue').

1.101 Command Reference : remres

```
   REMRes <resident pointer>
```

Remove a resident module from the 'resm' list.

'remres' uses autodefault to the 'resm' list for the first argument.

Example :

```
< remres exec <enter>
```

Related lists: resm

1.102 Command Reference : remstruct

```
                  REMStruct <struct def pointer>
```

Unload a structure definition previously loaded with
addstruct
.

'remstruct' uses autodefault to 'stru' for the first argument.

Related commands:
addstruct

interprete

view
Relate functions: peek() apeek() stsize()

Related lists: stru

Related tutor chapters: Looking at things

1.103 Command Reference : remtag

```
                  REMTag <address>
```

Remove the range of memory starting with <address> from the current tag
list. If the range does not exist this command does nothing.

Example :

```
< addtag 1000 50 as <enter>
< ...
< remtag 1000 <enter>


Related commands:
                 addtag

                 cleartags

                 loadtags

                 savetags

                 tags

                 usetag

                 tg

                 checktag

                 view
               Related functions:  taglist()

Related tutor chapters:  Looking at things
```

1.104 Command Reference : remvar

```
                  REMVar {<variables>}


This command removes PowerVisor variables if they exist.
'Remvar' is very useful in scripts. If a variable on the commandline
does not exist, 'remvar' will give NO error but simply continue with the
other variables on the commandline. You can only remove variables.
Functions, special variables and constants can't be removed with this
command. 'rc' and 'error' are also private and can't be removed.


Example :

< myvar=1000 <enter>
< remvar myvar <enter>

< disp myvar <enter>
> Addressed element not found !


Related commands:
                 vars

                 assign
               Related tutor chapters:  Scripts
```

1.105 Command Reference : reqload

```
                  INPUT,<ptr to filename> <- REQLoad <title>


Ask the user for a filename with a filerequester. This command uses
reqtools.library (V37 or higher) (© Nico François) if available, otherwise
this command is equivalent to the
                  scan
                  command.
The result of this command (in input), is a pointer to the file name
(or 0 if the user pressed 'cancel'). This string is remembered until you
quit PowerVisor or until you use another input command (
                  scan
                 ,
                  reqload
                 ,

                  reqsave
                  or
                  getstring
                 ).

This command returns a string if used from ARexx.

<title> is the title to be put in the requester window

The filerequester is a 'load' requester. This means that you can
double click on filenames.


Related commands:
                  reqsave

                  request

                  getstring

                  scan
```

1.106 Command Reference : reqsave

```
                  INPUT,<ptr to filename> <- REQSave <title>


Ask the user for a filename with a filerequester. This command uses
reqtools.library (V37 or higher) (© Nico François) if available, otherwise
this command is equivalent to the
                  scan
                  command.
The result of this command (in input), is a pointer to the file name
(or 0 if the user pressed 'cancel'). This string is remembered until you
quit PowerVisor or until you use another input command (
                  scan
```

```
                ,
                 reqload
                ,

                 reqsave
                 or
                 getstring
                ).

This command returns a string if used from ARexx.

<title> is the title to be put in the requester window

The filerequester is a 'save' requester. This means that you can't
double click on filenames.


Related commands:
                 reqload

                 request

                 getstring

                 scan
```

1.107 Command Reference : request

```
                   <result> <- REQuest <body string> <gadget string> <argument>


Show a requester on the PowerVisor screen. This command uses
reqtools.library (V37 or higher) (© Nico François) is available, otherwise
this command is simulated using the 'key' function.
The result of this command is a number indicating the button that was
pressed (this number is also what you should press on the keyboard if
you don't have reqtools.library). The rightmost button is 0. All other
buttons are numbered from left to right starting with 1.

<body string> is the string for the requester title bar.
<gadget string> is the string with all the button text below.
Note that you can use one % C-formatting character (optionally) in one
of the above strings. If you do this <argument> is used for the value.
Note that <argument> is NOT an optional argument.


Example :


< result={request 'Body : %ld' 'LeftBut|MidBut|RightBut' 1001} <enter>


Related commands:
                 getstring
```

reqload

reqsave
Related functions: key()

1.108 Command Reference : resident

RC,<Pointer to code> <- RESIdent [<filename>]

The 'resident' command loads a given file (with 'LoadSeg') and stores a pointer to the start of the program in 'RC'. You can use this pointer with the
go
command.

You can make ML scripts resident but you must make sure that the routines are pure.

If you give no argument to 'resident' you will get a list with all loaded code pointers.

Related commands:
unresident

go

script
Related tutor chapters: Scripts

1.109 Command Reference : rwin

RWin [<number of lines>]

Open/close the refresh window. (required if you want to use the
refresh
command).

Normally the height of the new logical window is 30 % of the total physical window height ('Main' is the physical window for all standard logical windows). However, if you specify <number of lines>, the logical window will be opened with <number of lines> visible lines (This is the number of lines when the default PowerVisor font is used).

By default the 'Refresh' logical window has the following characteristics :

- Number of columns is fixed and equal to the maximum number of columns visible at the time the logical window is created
- Number of rows is fixed and is always equal to 50
- -MORE- checking is disabled

```
   - Interrupt/Pause checking is disabled
   - Home position is real-top
   - Auto Output Snap is off

You can change these characteristics with the
                 prefs
                 command.

Note that if the 'intui' mode flag is one (this is off by default, see the

                 mode
                 command) the logical window will be opened on a new physical  ↩
                    window.
<number of lines> is ignored in that case.


Related commands:
                 refresh

                 dwin

                 swin

                 xwin

                 awin

                 owin

                 fit

                 colrow

                 prefs

                 mode
                Related lists:  lwin

Related tutor chapters:  Screens and windows
```

1.110 Command Reference : rx

```
                   RX <file name>


Execute an ARexx script. The default extension for ARexx scripts is 'pv'.
You do not need to type this extension unless you use a different one.

The name of the PowerVisor ARexx port is :

   REXX_POWERVISOR


Example :
```

```
   See the
                assign
                command


Related commands:
                script

                assign

                clip

                remclip
               Related tutor chapters:  Scripts
```

1.111 Command Reference : save

```
                  <bytes written> <- SAve <filename> <start> <bytes>


Save memory to disk.

This command displays the actual number of bytes written.


Example :

< save testfile 2000 1000 <enter>
> 000003E8,1000


Related commands:
                load
```

1.112 Command Reference : saveconfig

```
                  SAVEConfig


This commands saves the current
                mode
                and
                prefs
                settings to the file
s:PowerVisor-config. They will automatically be loaded when you start
PowerVisor.

This command also remembers the state of all the standard logical windows
(their position and if they are on a physical window or not) (Note that
this feature is new for PowerVisor V1.13). This means that you can simply
use the
                screen
```

command and the mouse to position all the standard
windows (see also the 'mode intui' option) and 'saveconfig' will save
the positions (Note that you still have to open them manually).

Related commands:
prefs

mode
Related tutor chapters: Installing PowerVisor

1.113 Command Reference : savetags

SAVETags <file> <base>

Save tags in the current tag list to a file.
<base> is simply saved in the file. You can use <base> later on when
you reload the tags to relocate them.

Related commands:
loadtags

addtag

remtag

cleartags

view

tg

usetag

checktag

tags
Related functions: taglist()

Related tutor chapters: Looking at things

1.114 Command Reference : scan

INPUT,<ptr to inputline> <- SCAn [<number>]

Ask the user for input. This command is very useful for ARexx scripts.
The result of this command (in input), is a pointer to the result string.
This string is remembered until you quit PowerVisor or until you use
another input command (

```
                scan
               ,
                reqload
               ,
                reqsave
                or
                getstring
               ).

This command returns a string if used from ARexx.

Normally the prompt is '????'. If you want another 4 letter prompt you
can use <number>.


Example :


< scan *"TEST" <enter>
TEST> This is a test <enter>

< memory input 20 <enter>
> 07EEBC82: 54686973 20697320 61207465 73740000              This is a test..
> 07EEBC92: 00000000                                        ....


Related commands:
                getstring

                reqload

                reqsave
               Related functions:  key()
```

1.115 Command Reference : screen

```
                  SCREen [<screen>]


Move the PowerVisor window to another screen. If you want PowerVisor on its
own screen, type 'screen' without arguments (this is default).
You can use the
                size
                command to change the size for this window (or you
can use the sizinggadget). You can use
                move
                to move the physical window.

'screen' uses autodefault to the 'scrs' list for the first argument.

If <screen> is 0 PowerVisor will move the 'Main' physical window on its own
screen, but the window will be resizable and movable. This is useful if
you want to customize your own design for PowerVisor.

If you want you can start with the PowerVisor window on the workbench
```

```
by default. Just type
                 saveconfig
                 when PowerVisor is on the WorkBench.
'saveconfig' will also remember the size of the window.


Example :

< screen workbench <enter>

move PowerVisor to workbench screen

< screen <enter>

back to own screen


Related commands:
                 size

                 move

                 prefs

                 saveconfig
                Related lists:  scrs   pwin

Related tutor chapters:  Screens and windows
```

1.116 Command Reference : script

```
                   <result> <- SCRIpt <script file> [<commandline>]


The 'script' command executes <script file>.
You can use comments in script files by preceding the line with ';'.

You can also execute machinelanguage scripts with this command.
These machinelanguage scripts are normal executable files. You can
write them in any language you wish as long as that language
has AmigaDOS executable files as output format. Although other
languages (like C) are possible we still call these scripts machinelanguage
scripts or ML scripts.

When a ML script is executed the registers contain the following info :

   a0=pointer to <commandline>
   a1=pointer to the 'rc' variable. (You can use this to return something
      to PowerVisor)
   a2=pointer to the PVCallTable.
   the value in d0 is returned to PowerVisor

<result> is the value returned in d0 (if ML script).

When you try to execute a script, PowerVisor will first try to execute
```

the script in the current directory, if that does not succeed, PowerVisor
will try the s:pv/ subdirectory. This is the recommended place for scripts.

Related commands:
rx

resident

unresident

go
Related tutor chapters: Scripts

1.117 Command Reference : scroll

SCROll <logical window> <x> <y>

Scroll the logical window. <x> and <y> is the new top-left visible
position. This command checks for illegal values.

'scroll' uses autodefault to the 'lwin' list for the first argument.

Related lists: lwin

Related tutor chapters: Screens and windows

1.118 Command Reference : search

RC,<address> <- Search <start> <bytes> <string>

Search to a string in memory starting at <start>. Search until the
string is found, or until you have searched <bytes> bytes.

Use the
next
command to continue the search.
You can use the lastfound() function to see where 'next' will continue
its search.

Example :

```
< a=alloc(n,1000) <enter>
< fill a+100 4 'test' <enter>

< search a 1000 'test' <enter>
> 07E83756 , 132659030
```

```
Will search for the string "test" beginning at 'a'. If after 1000 bytes
the string is not found, 0 is the result.


Related commands:
                 next

                 copy

                 fill
                Related functions:  lastfound()
```

1.119 Command Reference : setflags

```
                   RC,<oldflags> <- SETFLags <logical window> <mask> <flags>


Set flags for a logical window.
This command also displays the old flags and puts the result in 'rc'.
Bits in <mask> and <flags> :

   4   = -MORE- enabled (1 is enabled)
   32  = real-top (1 is real-top)
   64  = statusline off (1 is off)
   128 = interrupt off (1 is off)
   256 = auto output snap (1 if on)

Note that it is generally better to use
                 prefs
                 for the standard logical
windows instead of 'setflags'. This is because the value made with
'setflags' are reset everytime a new logical window is created or
another logical window disappears. The problem with 'prefs' is that you
first have to reopen the logical window before it has effect. If you use
both (you can make a script or alias for that) there is no problem.

'setflags' uses autodefault to the 'lwin' list for the first argument.


Example :

< setflags main 32+4 32 <enter>
> 00000106 , 262

will make the logical window real-top and disableds -MORE- checking.
262 is the old value.

To get the current value, type :

< setflags main 0 0 <enter>
> 00000122 , 290


Related commands:
                 openlw
```

```
                closelw

                awin

                owin

                rwin

                dwin

                swin

                xwin

                prefs
               Related lists:  lwin

Related tutor chapters:  Screens and windows
```

1.120 Command Reference : setfont

```
                  SETfont <logical window> <fontname> <fontheight>


Set a font for a logical window. The font must be either memory resident
or available in the 'fonts:' directory. Proportional fonts are not
supported.

Note that this command is only temporary and only for the contents
of a logical window (the text in the logical window).
If you want another default font for all PowerVisor logical windows,
string gadget, size bars, ... you must use 'prefs font'.

'setfont' uses autodefault to the 'lwin' list for the first argument.

There are seven standard logical windows in the current release of
PowerVisor, (you can open more if you want) :

   Rexx     : rexx output window
   PPrint   : PortPrint output window
   Refresh  : refresh window
   Debug    : debugging window
   Extra    : extra window
   Main     : main window
   Source   : source window for source level debugger

See the 'lwin' list for all available logical windows.
All these logical windows can use another font.


Example :

If the 'Extra' logical window is not already open, open it with :
```

```
< xwin <enter>

< setfont extra topaz.font 9 <enter>

Try the new font :

< on extra list task <enter>
> ... (in topaz 9)


Related commands:
                 xwin

                 dwin

                 swin

                 rwin

                 awin

                 owin

                 openlw

                 closelw
                Related lists:  lwin

Related tutor chapters:  Screens and Windows
```

1.121 Command Reference : showalloc

```
                   SHowalloc


This command shows all memory blocks allocated with the  alloc()
function and the
                 rblock
                 command. All memory blocks that
are in this list will be automatically freed when PowerVisor quits or when
you execute the
                 cleanup
                 command.


Example :

< a=alloc(n,1000) <enter>
< b=alloc(n,100) <enter>

< showalloc <enter>
> 07EA6912,132802834
> 07EA77D2,132806610

< cleanup <enter>
```

```
< showalloc <enter>


Related commands:
                cleanup

                rblock
               Related functions:  alloc()   free()   getsize()   isalloc()    ↩
                  realloc()
```

1.122 Command Reference : size

```
                  SIze <physical window> <x> <y>


Set the x and the y size (in pixels) for the physical window.
This command only works when the physical window is resizable (not a
backdrop window).
You can move PowerVisor to another screen with the
                screen
                command.
You can of course also size the PowerVisor window using the size gadget.

'size' uses autodefault to the 'pwin' list for the first argument.


Related commands:
                screen

                move
               Related lists:  pwin

Related tutor chapters:  Screens and windows
```

1.123 Command Reference : source

```
                  SOurce 'l' <filename> [<hunkaddress>] | 'w' <address> | 's' |
         't' <tab size> | 'r' | 'c' | 'g' <line>


Control the source for the current debugtask. Note that the source
for the current debug task is always displayed in the 'Source' logical
window. You must compile or assemble your program with debug hunk
information if you want to use this feature. PowerVisor currently
understands the debug hunk format used by 'Macro68' (include the 'debug'
statement in your source) and by 'SAS/C' (compile with '-d1', do not
use any number other than 1!). This format is also compatible with the
debug hunk format used by Devpac3. The 'source' command could work for
other compilers too but this is not tested.

Note that if you have loaded the source for the current debug task you
can use the linenumber operator '#'.
```

```
   source l <filename> [<hunkaddress>]
               Load the source for the current debug task.
               If you give <hunkaddress>, PowerVisor will load the source
               for the given hunks. This is extremely useful when you have
               created a dummy debug task.
               Note that <hunkaddress> is 4 more than the number given in
               the hunklist with the
                 hunks
                 command.
               Note that <hunkaddress> is not optional when you are loading
               the source for a dummy debug task
   source w <address>
               Use this command to see in which source file and on which
               line a specific address is located
   source t <tab size>
               Set the tab size used for the source display. The default
               tab value is 8
   source s
               Show all sources for the current debug task
   source r
               Redisplay the source in the 'Source' logical window
   source c
               Clear all sources and unload them
   source g <line>
               Goto a line in the current loaded source. Note that you can
               also scroll in the current file with the mouse. Simply press
               the left mouse button in the first or the last line of the
               'Source' logical window


Related commands:
                 debug

                 trace

                 break

                 duse

                 with

                 symbol

                 swin
               Related functions:  getdebug()

Related lists:  dbug

Related tutor chapters:  Debugging
```

1.124 Command Reference : specregs

```
                  SPecregs


(only 68020, 68030 or 68040)
Show all special 68020 registers.


Example :

< specregs <enter>
> MSP  : 560F5B16
> ISP  : 07E02228
> USP  : 07E8BC64
> SFC  : 00000007
> DFC  : 00000007
> VBR  : 00000000
> CACR : 00002111
>    Write Allocate set
>    Disable Data Burst
>    Clear Data Cache not set
>    Clear Entry in Data Cache not set
>    Freeze Data Cache not set
>    Enable Data Cache
>    Enable Instruction Burst
>    Clear Instruction Cache not set
>    Clear Entry in Instruction Cache not set
>    Freeze Instruction Cache not set
>    Enable Instruction Cache
> CAAR : B8F77BED


Related commands:
                 mmutree

                 mmuregs

                 mmurtest

                 mmuwtest
                Related tutor chapters:  Looking at things
```

1.125 Command Reference : speek

```
                  <value> <- SPEEk <address>


(only 68030 or 68040)
Peek a longword from memory. <address> is a PHYSICAL address. The MMU tree
is completely ignored.


Example :

< speek 4 <enter>
> 07E007CC , 132122572
```

```
Related commands:
                 mmutree

                 spoke
               Related tutor chapters:  Looking at things
```

1.126 Command Reference : spoke

```
                   SPOke <address> <value>


(only 68030 or 68040)
Poke a longword in memory. <address> is a PHYSICAL address. The MMU tree
is completely ignored.


Related commands:
                 mmutree

                 speek
               Related tutor chapters:  Looking at things
```

1.127 Command Reference : sprint

```
                   SPRint <string>


Print a string on a serial terminal (Useful for the
                 addfunc
                 command, for
example).


Example :

< sprint 'First : \(1+1000), Then : \(65,%lc)\0a' <enter>
SERIAL> First : 1001, Then : A

< sprint test <enter>
SERIAL> test


Related commands:
                 addfunc
```

1.128 Command Reference : stack

```
                  STAck [<task> <micro seconds>]


Use this command to check stack usage for a specific task. You can only
check one task at the time. With no arguments this command removes the
stack checker.

'stack' uses the UNIT_MICROHZ timer device. This means that <micro seconds>
must be greater or equal than 2. This command has a better check resolution
than 'account' (if your <micro seconds> value is low enough).

'stack' freezes the task when a stack overflow is about to occur (see the

                 prefs
                 command for the minimum number of bytes allowed in the stack).
Note that this behaviour is different from
                 account
                . Stack overflows
trapped with the 'account' stack checker are put in the 'crsh' (crash)
list (This behaviour may change in future).

In addition to stack checking this command also computes the maximum stack
usage for the task. You can display this maximum with the  getstack()
function.

'stack' uses autodefault to the 'task' list for the first argument.


Related commands:
                 prefs

                 stack
                Related functions:  getstack()

Related lists:  crsh
```

1.129 Command Reference : string

```
                  <string> <- STRIng <string pointer>


This command simply evaluates its argument and returns it. This command is
useful in ARexx if you want to convert a string pointer (a pointer to a
string) to an ARexx string. This is because this function returns a string
when used in ARexx


Related commands:
                 assign
                Related tutor chapters:  Scripts
```

1.130 Command Reference : swin

```
                  SWin [<number of lines>]


Open/close the source window. (required if you want to use
sourcelevel debugging).

Normally the height of the new logical window is 30 % of the total
physical window height ('Main' is the physical window for all standard
logical windows). However, if you specify <number of lines>, the logical
window will be opened with <number of lines> visible lines (This is
the number of lines when the default PowerVisor font is used).

By default the 'Source' logical window has the following characteristics :

   - Number of columns is autoscalable (-1)
   - Number of rows is autoscalable (-1)
   - -MORE- checking is disabled
   - Interrupt/Pause checking is disabled
   - Home position is real-top
   - Auto Output Snap is off

You can change these characteristics with the
                prefs
                command.

You can't use
                on
                with the 'Source' logical window.

Note that if the 'intui' mode flag is one (this is off by default, see the

                mode
                command) the logical window will be opened on a new physical  ↩
                   window.
<number of lines> is ignored in that case.


Related commands:
                rwin

                dwin

                xwin

                awin

                owin

                debug

                prefs

                source
```

```
                mode

                on
               Related lists:  lwin

Related tutor chapters:  Screens and windows   Debugging
```

1.131 Command Reference : symbol

```
                  SYmbol 'l' <filename> [<hunkaddress>] | 'c' | 'a' <symbol> < ↩
                     value> |
         'r' <symbol> | 's'


Control the symboltable for the current debugtask.

   symbol l <filename> [<hunkaddress>]
               Load the symbols for the current debug task.
               If you give <hunkaddress>, PowerVisor will load the symbols
               for the given hunks. This is extremely useful when you have
               created a dummy debug task.
               Note that <hunkaddress> is 4 more than the number given in
               the hunklist with the
                 hunks
                 command.
               Note that <hunkaddress> is not optional when you are loading
               symbols for a dummy debug task.
   symbol c    Clear all symbols for the current debug task.
   symbol a <symbol> <value>
               Add a symbol with a specified value the the symbol list
               for the current debug task.
   symbol r <symbol>
               Remove a symbol from the symbol list for the current
               debug task.
   symbol s    List all symbols for the current debug task.


Related commands:
                debug

                trace

                break

                duse

                with

                source
               Related functions:  getdebug()

Related lists:  dbug

Related tutor chapters:  Debugging
```

1.132 Command Reference : sync

```
                  SYNc


Use this command to synchronize PowerVisor with ARexx. Normally you can
execute an ARexx script (with
                rx
               ) and while this script is executing
you can still use PowerVisor for other things. When you use this command,
PowerVisor will disable user input. Only ARexx commands are accepted.

Do not forget to
                async
                in your ARexx script.


Related commands:
                async

                rx
               Related tutor chapters:  Scripts
```

1.133 Command Reference : tags

```
                  TAGs


List all tags in the current tag list.


Related commands:
                addtag

                remtag

                cleartags

                loadtags

                savetags

                usetag

                tg

                checktag

                view
               Related functions:  taglist()

Related tutor chapters:  Looking at things
```

1.134 Command Reference : taskpri

```
   TASKPri <task ptr> <priority>
```

Set the priority for a task.

'taskpri' uses autodefault to the 'task' list for the first argument.

Example:

```
< taskpri task:test -5 <enter>
```

Related lists: task

1.135 Command Reference : tg

```
                  TG <number> <command>
```

Temporarily set the current tag list to <number> and execute the command.
<number> is in 0 .. 15.

Related commands:
usetag

addtag

remtag

tags

savetags

loadtags

view
Related functions: taglist()

Related tutor chapters: Looking at things

1.136 Command Reference : to

```
                  TO <file> <command>
```

Log the output of one command (or group) to a file. After the command
has executed the previous log file (if any) is reinstalled.

```
Examples :

< to ram:MyOutputFile list task <enter>
> ...

The output will still appear on the current logical window. This command
temporarily works like the
                log
                command. The real log file is restored
after this command exits.
If you only want output in a file you can use :

< -to ram:MyOutputFile list task <enter>

or

< to ram:MyOutputFile -list task <enter>

You can also combine the 'to' and the
                on
                command :

First open the 'Extra' window (if it is not already open) :
< xwin <enter>

< to ram:MyOutputFile on extra list task <enter>
> ...

This command will list all tasks on the 'Extra' logical window. No output
will be written to the file since the 'to' command only redirects the
output from the current logical window. However :

< on extra to ram:MyOutputFile list task <enter>
> ...

will also list all task on the extra window. The difference is that this
time there will be output in the file since the 'to' command redirects
the output from the current logical window. At the time the 'to' command
is executed, this logical window is equal to 'Extra'.


Related commands:
                log

                xwin

                on
               Related lists:  lwin

Related tutor chapters:  Screens and windows
```

1.137 Command Reference : trace

```
                  TRace [ 'n' <number> | 'b' | 'r' <register> | 'u'['t'] <address ↩
                     >
         | 'o'['t'] | 'c' <condition> | 's' | 'g'['t'] | 'h' | 'f'
         | 't' | 'j' ]


Use this command to singlestep a program (Use
                debug
                first to make
a debug task).

   trace n <number>        Trace <number> instructions. Tracing is done
                           in singlestep mode.
   trace b                 Trace until the next branch (singlestep mode).
   trace t                 Skip BSR or JSR. If not a BSR or JSR simple
                           trace is used. This function works in ROM.
                           Execute mode is used.
   trace j                 Trace until use of library ROM function:
                           JSR or JMP (a6).
                           Tracing is done in singlestep mode.
   trace r <register>      Trace until a specified register is changed
                           (singlestep mode).
                           register can be d0-d7, a0-a6 or sp.
   trace u <address>       Trace until the programcounter is equal to
                           <address>. Tracing is done in execute mode.
   trace ut <address>      Same as above but use singlestep mode instead.
   trace o                 Trace over the current instruction. Tracing
                           is done in execute mode.
   trace ot                Same as above but use singlestep mode instead.
   trace c <condition>     Trace until a condition is satisfied. This
                           condition is a string. Example:
                           trace c '@d1==@d2' will trace until register
                           d1 is equal to register d2. Tracing is done
                           in singlestep mode.
   trace s                 Skip an instruction
   trace i                 Do not trace. Simply show the current registers
                           and instructions.
   trace g                 Execute until a breakpoint is encountered
   trace gt                Same as above but tracing is done in singlestep
                           mode.
   trace h                 Interrupt the tracing or executing of the
                           current debug task.
   trace f                 Interrupt the tracing or executing of the
                           current debug task as soon as this task is
                           in ready state.


Related commands:
                debug

                break

                duse

                drefresh

                dstart
```

```
                dscroll

                dwin

                swin

                with

                symbol
               Related functions:  getdebug()

Related lists:  dbug

Related tutor chapters:  Debugging
```

1.138 Command Reference : track

```
                  TRACK 't' <task> | 's' | 'c' | 'l'


With this command you can control the resource tracker. This resource
tracker remembers all memory allocations (AllocMem, AllocVec) and
all opened libraries (OpenLibrary, OldOpenLibrary). With this information
you can see if the task (or process) cleans everything up.
Note that you can only track resources for one task or process at a
time.

'track' uses autodefault to the 'task' list for the <task> argument.

   track t     Take a task or process in the task list and start resource
               tracking for that task
   track s     Stop tracking and free all track information. Note that
               the memory that the task forgot to free is NOT freed with
               this function
   track c     Stop tracking and free all track information (like
               'track s')
               In addition this function also clears all memory that is
               not freed by the program
   track l     List all memory and libraries current allocated or open


Example :

The easiest way to start resource tracking for a program is to use
'debug n', start the program, use 'track t' to get the new task from the
task list and use 'debug r' to remove the debug node and let the program
continue.

< track l <enter>
> 07F3211A : 07E0E1F0,00000000 dos.library
> 07F3217C : 07EA5EE8,00000010
> 07F33CAC : 07E5B8A0,00000000 mathieeedoubbas.library
> 07F33CCC : 07EBA8A8,00000000 mathieeedoubtrans.library
> 07F32ACA : 07E0ABC4,00000024 intuition.library
```

```
> 07F32ADC : 07E036B0,00000024 graphics.library
> 07F32AEE : 07E49820,00000024 gadtools.library
> 07F32B00 : 07E00154,00000024 utility.library
> 07F32B12 : 07E389E0,00000026 locale.library
> 00F81CE8 : 07EBA8E8,000000AC
> 00F92B5C : 07EBA8EC,000000A8 (AllocVec)
> 00F81CE8 : 07EBA998,0000001C
> 00FA7358 : 07EBA99C,00000018 (AllocVec)
> 00FD8F90 : 07F2D0A8,000000A4
> ...

The first hexadecimal number is the program counter where the library
function (AllocMem, AllocVec, OpenLibrary or OldOpenLibrary) was called.
The second number is the result of this library function (either the
allocated memory or the opened library). The last number is the version
of the library or the size of the allocated memory block. If it is a
library the name of the library is printed at the end. If the memory is
allocated with 'AllocVec' (AmigaDOS 2.0 only) '(AllocVec)' is printed.


Related commands:
                 debug
                Related lists:  task
```

1.139 Command Reference : unalias

```
                   UNALias <alias command name>


Remove an alias string previously installed with
                 alias
                .


Example :


< alias xxx 'disp 3' <enter>
< xxx <enter>
> 00000003 , 3

< unalias xxx <enter>
< xxx <enter>
> Syntax Error !


Related commands:
                 alias
                Related tutor chapters:  Installing PowerVisor
```

1.140 Command Reference : unasm

```
                  Unasm [<start> [<instructions>]]


This command disassembles machinelanguage beginning at <start>.
<instructions> instructions are disassembled.
This command also shows symbols and breakpoints if there are any.
The disassembler supports the following processors :
   68000, 68010, 68020, 68030, 68040, 68881, 68882, 68851

With no arguments this command continues the disassembly.

<instructions> is 20 by default.
PowerVisor remembers the last number of lines used with this command
and uses this number as the new default number of <instructions> for
the following 'unasm' commands.

Normally you will also see the hex data corresponding with the code.
You can disable this (and thus provide for longer labels in symbolic
disassembly) with the
                mode
                command (noshex).

If you press <enter> after an 'unasm' command, the disassembly will
continue.


Related commands:
                view

                memory

                mode
               Related functions:  lastmem()   lastlines()

Related tutor chapters:  Looking at things
```

1.141 Command Reference : unfreeze

```
                  UNFreeze <task>


Unfreeze a task you have frozen before.

'unfreeze' uses autodefault to the 'task' list for the first argument.


Related commands:
                freeze

                kill
               Related lists:  task
```

1.142 Command Reference : unhide

```
                  UNHide


Unhide all output from commands issued from an ARexx script.


Related commands:
                 hide

                 rx
                Related tutor chapters:  Scripts
```

1.143 Command Reference : unloadfd

```
                  UNLoadfd <fd-file node>


Remove all functiondefinitions in a fd-file. You can find the
nodes for fd-files in the 'FDFi' list.
After this command you cannot use the library functions from this
fd-file anymore.

'unloadfd' uses autodefault to the 'FdFi' list for the first argument.


Related commands:
                 loadfd

                 libfunc

                 libinfo
                Related lists:  fdfi
```

1.144 Command Reference : unlock

```
                  UNLOCk <pointer to a lock>


This command unlocks a lock. The pointer to the lock must be an APTR,
not a BPTR !


Related commands:
                 pathname
                Related lists:  lock   fils
```

1.145 Command Reference : unresident

```
                  UNResident <Pointer to code>


Unload a file loaded with
                 resident
                . All resident files are automatically
unloaded when PowerVisor quits.


Related commands:
                 resident

                 go

                 script
                Related tutor chapters:  Scripts
```

1.146 Command Reference : usetag

```
                  USetag <number>


Set the current tag list to <number> (0 .. 15). The current tag list is
used by all tag commands and by 'view'. The default current tag list
is 0.

You can also use
                 tg
                 to temporarily set the current tag list.


Related commands:
                 tg

                 addtag

                 remtag

                 tags

                 savetags

                 loadtags

                 view
                Related functions:  taglist()

Related tutor chapters:  Looking at things
```

1.147 Command Reference : vars

```
                    VArs ['all']


This command shows all variables. You will notice the variables
'rc' and 'error'.
When you use the 'all' option, PowerVisor will show all variables
including constants and functions.
Notice the 'mode' special variable, the 'version' constant and
the 'input' constant.

You can type really anything for the 'all' argument. The simple presence
of an extra argument is enough.


Example :

< a=1000 <enter>

< vars <enter>
> error                                   : 0   00000026 , 38
> rc                                      : 0   07E43080 , 132395136
> a                                       : 0   000003E8 , 1000

< vars all <enter>
> mode                                    : 2   00002159 , 8537
> version                                 : 1   00000200 , 512
> error                                   : 0   00000026 , 38
> rc                                      : 0   07E43080 , 132395136
> input                                   : 1   00000000 , 0
> getx                                    : 3   07E7F1D6 , 132641238
> gety                                    : 3   07E7F1DE , 132641246
> getdebug                                : 3   07E86992 , 132671890
> getchar                                 : 3   07E7F1E6 , 132641254
> base                                    : 3   07E80562 , 132646242
> rp                                      : 3   07E7F224 , 132641316
> lines                                   : 3   07E7F1FC , 132641276
> cols                                    : 3   07E7F210 , 132641296
> key                                     : 3   07E7F2E4 , 132641508
> alloc                                   : 3   07E7D586 , 132633990
> free                                    : 3   07E7D5C8 , 132634056
> getsize                                 : 3   07E7D5EA , 132634090
> realloc                                 : 3   07E7D604 , 132634116
> lastmem                                 : 3   07E7D55A , 132633946
> lastfound                               : 3   07E7D562 , 132633954
> peek                                    : 3   07E804EC , 132646124
> apeek                                   : 3   07E80526 , 132646182
> rfrate                                  : 3   07E78310 , 132612880
> rfcmd                                   : 3   07E78318 , 132612888
> isalloc                                 : 3   07E7D5B0 , 132634032
> curlist                                 : 3   07E80164 , 132645220
> gethist                                 : 3   07E78170 , 132612464
> qual                                    : 3   07E7F2DC , 132641500
> getcol                                  : 3   07E7EF9C , 132640668
> getrow                                  : 3   07E7EFB0 , 132640688
> getlwin                                 : 3   07E7EF6E , 132640622
> stsize                                  : 3   07E804D8 , 132646104
```

```
> taglist                                   : 3   07E7CD22 , 132631842
> toppc                                     : 3   07E8584C , 132667468
> botpc                                     : 3   07E85868 , 132667496
> eval                                      : 3   07E8446C , 132662380
> if                                        : 3   07E84452 , 132662354
> isbreak                                   : 3   07E85818 , 132667416
> a                                         : 0   000003E8 , 1000

The first argument after the colon means the following :

   0 = normal variable
   1 = constant variable
   2 = special variable with an action attached to it (internal)
   3 = function (value is pointer to code for function)


Related commands:
                 mode

                 error

                 remvar

                 assign
```

1.148 Command Reference : view

```
                   VIew [<address> [<bytes>]]


View memory following certain requirements. The requirements follow from
the current taglist (which is set with
                 usetag
                 or
                 tg
                ). The default
format is Long/Ascii combined view. You can change the format used to
display the memory for a specific range of memory to something else using
tags. The default number of bytes to view is 320 (16 bytes * 20 lines). If
you do not give an address, PowerVisor will continue the listing of
memory where you stopped in a previous listing (either
                 view
                ,
                 memory
                or
                 unasm
                ). Pressing <Enter> after a 'view' command also causes
the listing to continue.

PowerVisor remembers the last number of bytes used with the command
(and the 'memory' command) and uses this number as the new default
number of <bytes> for the following 'view' (or 'memory') commands.

Use the
                 addtag
```

command to define how a specific range of memory should be viewed.

Also see the Looking at things chapter for more examples and details about these tags.

Example :

First we define the memory starting on location 0 as a range of longwords :

```
< addtag 0 50 la <enter>
```

This 'addtag' command adds a definition for a range of memory. A memory range with 50 bytes starting from address 0 is defined as LA. This is Long/Ascii. This is the default, so you won't see anything special when you view that memory.

```
< addtag 50 50 wa <enter>
```

The next 50 bytes of memory (starting on address 50) are defined as WA or Word/Ascii. We can use the 'view' command to see what we have done :

```
< view 0 <enter>
```

(Note that the 'view' command has the same sort of arguments as the 'memory' command).

```
> 00000000: 00000000 07E007CC 00F80834 00F80B16              ...........4....
> 00000010: 00F80ADA 00F80ADC 00F80ADE 00F80AE0              ................
> 00000020: 00F80C00 00F80AE4 00F80AE7 00F80AE8              ................
> 00000030: 00F8                                             ..
> 00000032: 0AEA 00F8 0AEC 00F8 0AEE 00F8 0AF0 00F8          ................
> 00000042: 0AF2 00F8 0AF4 00F8 0AF6 00F8 0AF8 00F8          ................
> 00000052: 0AFA 00F8 0AFC 00F8 0AFE 00F8 0B00 00F8          ................
> 00000062: 0B02                                             ..
> 00000064: 00F810F4 00F81152 00F81188 00F811E6              .......R........
> 00000074: 00F8127C 00F812C6 00F81310 00F80B70              ...|...........p
> 00000084: 00F80B72 00F80B74 00F80B76 00F80B78              ...r...t...v...x
> 00000094: 00F80B7A 00F80B7C 00F80B7E 00F80B80              ...z...|...~....
> 000000A4: 00F80B82 00F80B84 00F80B86 00F80B88              ................
> 000000B4: 00F80B8A 00F80B8C 00F80B8E 00F80B90              ................
> 000000C4: 00F80B92 00F80B94 00F80B96 00F80B98              ................
> 000000D4: 00F80B9A 00F80B9C 00F80B9E 00F80BA0              ................
> 000000E4: 00F80BA2 00F80BA4 00F80BA6 00F80BA8              ................
> 000000F4: 00F80BAA 00F80BAC 00F80BAE 00000000              ................
> 00000104: 00000000 00000000 00000000 00000000              ................
> 00000114: 00000000 00000000 00000000 00000000              ................
> 00000124: 00000000 00000000 00000000 00000000              ................
> 00000134: 00000000 00000000 00000000                       ............
```

You can see that the memory starting at location 50 is listed in Word/Ascii format.

Related commands:

addtag

```
                remtag

                cleartags

                loadtags

                savetags

                tg

                usetag

                checktag

                tags

                memory

                unasm

                interprete
               Related functions:  taglist()   stsize()   lastmem()   lastbytes()

Related tutor chapters:  Looking at things
```

1.149 Command Reference : void

```
                  <result> <- Void {<argument> ...}


This command does absolutely nothing except evaluating it's arguments.
<result> is the result of the last evaluated expression.
Note that this command is useful in the following situations :

   - calling library functions or PowerVisor functions when you are not
     interested in the result of this function
   - returning a value (<result>) from a group operator :
         < disp {a=4;b=5;void a+b} <enter>
         > 00000009 , 9
   - you can also use the 'void' command to copy a PowerVisor expression
     to an ARexx variable :
         ARexx< options results
         ARexx< 'void a+4'
         ARexx< b=result


Related commands:
                disp
               Related functions:  if()   eval()

Related tutor chapters:  Getting Started   Expressions
```

1.150 Command Reference : wblock

```
                  WBlock <Unit number> <block number> <address>


Write a block to disk. <address> must be in chip ram.
A block is 1024 bytes big.


Related commands:
                rblock
```

1.151 Command Reference : with

```
                  WITh <debug node> <command>


This command temporarily sets another current debug task and executes
<command>. This is useful if you want to view memory with another set
of symbols for example. Or if you want to change the registers of another
debug task.

'with' uses autodefault to the 'dbug' list for the first argument.


Example:

< with MyBuggyProgram memory 070540 <enter>
> ...


Related commands:
                duse

                debug

                break

                trace

                drefresh

                symbol
               Related functions:  getdebug()

Related lists:  dbug

Related tutor chapters:  Debugging
```

1.152 Command Reference : xwin

```
                    Xwin [<numer of lines>]


Open/close the extra window. You can use the 'Extra' logical window for
everything you like. You can use it as a scratchbook for example. Using
the
                  on
                  command you can dump some information there.

Normally the height of the new logical window is 30 % of the total
physical window height ('Main' is the physical window for all standard
logical windows). However, if you specify <number of lines>, the logical
window will be opened with <number of lines> visible lines (This is
the number of lines when the default PowerVisor font is used).

By default the 'Extra' logical window has the following characteristics :

   - Number of columns is fixed and equal to the maximum number of columns
     visible at the time the logical window is created
   - Number of rows is fixed and equal to the maximum number of rows
     visible at the time the logical window is created
   - -MORE- checking is disabled
   - Interrupt/Pause checking is enabled
   - Home position is top-visible
   - Auto Output Snap is on

You can change these characteristics with the
                  prefs
                  command.

Note that if the 'intui' mode flag is one (this is off by default, see the

                  mode
                  command) the logical window will be opened on a new physical  ↩
                     window.
<number of lines> is ignored in that case.


Related commands:
                  rwin

                  dwin

                  swin

                  awin

                  owin

                  fit

                  colrow

                  prefs

                  mode
```

Related lists: lwin

Related tutor chapters: Screens and windows