TheWizCorner.hyper

COLLABORATORS

	TITLE : TheWizCorner.hyper	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		December 6, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

TheWizCorner.hyper

1.1 The Wizard Corner (Tue Nov 3 15:03:32 1992)

```
Contents:
   Introduction
   The 'pvcall' command
   Description of internal memory formats
   The bases

All bases:
   main base (pvcall 36)
   eval base (pvcall 30)
   arexx base (pvcall 31)
   debug base (pvcall 32)
   file base (pvcall 33)
   general base (pvcall 34)
   list base (pvcall 35)
   screen base (pvcall 40)
   memory base (pvcall 41)

Various:
   Back to main contents
```

1.2 The Wizard Corner : Introduction

```
WARNING !!! ONLY READ IF YOU THINK YOU ARE VERY EXPERIENCED WITH
POWERVISOR!

READ THE ABOVE STATEMENT AGAIN! (BUT DON'T GO IN AN INFINITE LOOP)


This file contains all information for the experienced script writer.
It explains the powerful  pvcall  command and lists the contents of
some internal PowerVisor data structures. With the information contained
in this chapter in combination with the  Scripts  chapter you can make very
powerful scripts. Some examples are given in the 'Source' subdirectory.

Note that there are include files (both .h and .i) for all
```

structures given in this file. These include files can be found in the PVDevelop/include/PV subdirectory.

For SAS/C users there is also 'PVCallStub.lib'. Using this library you can more easily call the pvcall routines from C. You can find this library in PVDevelop/lib.

If you want some examples you can look in the 'Source' subdirectory. This directory contains sources in C and machinelanguage using the PVCallTable and other internal variables.

1.3 The Wizard Corner : The 'pvcall' command

The pvcall command can be used to access internal variables and to install some extra features. The first argument of 'pvcall' is the number of the function you want to use (see below for a list of all functions). After this number follow the extra arguments (if any). Note that not all 'pvcall' functions are callable from within PowerVisor. Some are only intended to be called from within a machinelanguage script. To call a 'pvcall' function from within a machinelanguage script you can use the PVCallTable (offset 34 in the PowerVisor-port). This is a pointer to the table containing all pointers to the 'pvcall' functions. The pointer to the PVCallTable is also automatically available if your machinelanguage routine is executed from within PowerVisor (in register a2) (see the Scripts chapter for more information about machinelanguage scripts). The return value from these functions is always in d0. If a certain pvcall function is only available from machinelanguage an asterix ('*') is put after the number. All functions callable from both machinelanguage and PowerVisor expect their arguments on a commandline. If you want to use any of these functions in machinelanguage you have to build a commandline and provide the pointer to it in a0. Except for the 'Install<xxx>Cmd' functions ('Pre' command, 'Post' command, 'Quit' command and 'Snap' command) all functions preserve registers d2-d7 and a2-a6. The 'Instal<xxx>Cmd' functions preserve d2-d7 and a3-a6.

Number	Function
0	Create a new PowerVisor function <name> <address of routine> this is a pointer to a machinelanguage routine. When this routine is called, a0 points to the arguments and a2 points to the 'PVCallTable'.
1	Generate an error <error number>
2	Advance history buffer one line. Nothing happens if this line is the last. This function updates 252:4 in MainBase.
3	Lower history buffer one line. Nothing happens if this line is the first. This function updates 252:4 in MainBase.

```
4       Get current history line and copy to stringgadget buffer.
        The current history line is the line pointed to by 252:4 in
        MainBase. If 252:4 is 0 the stringgadget buffer is cleared.

5       Refresh the stringgadget. Use this function after you have
        changed something in the stringgadget buffer.

6       Install a 'Pre' command. This is a command that is executed
        before the commandline is parsed that is just typed in by
        the user. The command can find the pointer to the commandline
        in 'ScreenBase' (the stringgagdet buffer) and can make changes.
        See the  Technical information  chapter for the exact moment
        of the execution of this command. When you generate an error in
        the 'Pre' command, you will prevent further execution (The user
        can override both the 'Pre' and 'Post' commands with the '\'
        prefix commandline operator).
           <commandstring>

7*      Evaluate an expression.
        All expression features that PowerVisor supports are supported
        by this function (even groups).
           <a0 = pointer to expression>
           -> <a0 = pointer after expression or 0 if error (flags)>
           -> <d0 = resulting value>

8       Remove a variable, special variable, constant or function.
        Be careful with this command since you can remove internal
        variables like 'rc', 'mode' and 'error' with this function.
        Removing these variables will certainly do no good.
           <name>

9*      Parse a string from the commandline.
        The '\' and '·' (strong quote) operators are supported.
        Note that you are NOT responsible for freeing the string.
        The string is automatically added to the autoclear list.
        This also means that you must copy the string if you want
        to remember it permanently.
           <a0 = commandline>
           -> <a0 = pointer after string or 0 if error (flags)>
           -> <d0 = pointer to string>

10      Copy string to the stringgadget buffer
           <string>

11      Add string to the history buffer. Note that the PowerVisor
        history buffer never contains two equal history lines after
        each other. This function checks if the previous history line
        is equal to the one you are going to add. If they are equal
        nothing happens
           <string>

12      Get address of the stringgadget buffer
           -> <address>

13      Append string to the stringgadget buffer
           <string>
```

```
14      Skip spaces. This command skips all spaces in a string (','
        chars are also considered spaces).
           <string>
           -> <pointer to first non space character>

15      Set cursor position in stringgadget. Use this command to set
        the cursor position where it must be the next time a 'Scan' is
        executed. The internal 'Scan' routine is called to get the
        commandline and for the  scan  command.
           <position>

16      Install a 'Post' command. This is a command that is executed
        after the commandline is parsed and executed that was typed
        in by the user.
           <commandstring>

17      Set debug mode for PowerVisor.
        When debug mode is on, PowerVisor prints each command before
        it is executed (after alias expansion) and also prints
        the return code of each command. This is useful for debugging
        recursive aliases, scripts, macros and other special
        things.
           <debug number> = 0 for no debug, 1 for debugging info

18      Get execution level
           -> <execlevel>
                 0  = commandline
                 1  = script
                 2  = attach (IDC)
                 3  =  for  command
                 4  =  to  command
                 5  =  with  command
                 6  =  tg  command
                 7  =  on  command
                 8  = refresh
                 9  = group command
                 10 = snap command
                 11 = intuition handler command
                 12 = quit handler command
                 13 = signal handler command
                 14 = OBSOLETE
                 15 = called from 'ExecCommand' portprint function
                 16 = Pre command
                 17 = Post command

19      OBSOLETE

20      Get mStringInfo. Note that when you change something in this
        structure, you will probably have to call 'pvcall 52' or
        'pv 5' to remake the stringgadget.
           -> <pointer to StringInfo>
              Fields in the StringInfo structure.
                 offs  size  function
                 ---------------------------------
                 0     4     Buffer
                 4     4     UndoBuffer
                 8     2     BufferPos
```

```
                10    2     MaxChars
                12    2     DispPos
                14    2     UndoPos
                16    2     NumChars
                18    2     DispCount
                20    2     CLeft
                22    2     CTop
                24    4     LayerPtr
                28    4     LongInt
                32    4     AltKeyMap

21      Get Snap Buffer. This buffer is 120 bytes long.
           -> <pointer to snap buffer>

22      Install command before 'snap'. Using 'pvcall 21' you can change
        something in the string that is snapped. If you return 0 from
        this command the 'snap' will not happen.
           <commandstring>

23      OBSOLETE

24      Beep
           <period>
           <time>

25      Get address of variable or function
           <variable name>
           -> <address or null if it does not exists>

26      OBSOLETE (Since V1.10)

27      Create constant
           <name>
           <value>

28      Compare two strings
           <pointer to string 1>
           <pointer to string 2>
           <length>
           -> -1 if equal

29      Call machinelanguage script
           <pointer>
              Routine is called with a0 the pointer to the rest of the
              commandline, a1 the pointer to the RC variable, a2 the
              pointer to the PVCallTable and d6 equal to 0 (d6 is
              normally the number of arguments but 'pvcall 29' does not
              allow you to give arguments to the routine).
              The result of this 'pvcall' is the result from the
              routine in d0.

30      EvalBase
           -> <EvalBase>

31      ARexxBase
           -> <ARexxBase>
```

```
32       DebugBase
            -> <DebugBase>

33       FileBase
            -> <FileBase>

34       GeneralBase
            -> <GeneralBase>

35       ListBase
            -> <ListBase>

36       MainBase
            -> <MainBase>

37       Routines. You may change this routine table but if you do so
         you must make sure that the list remains sorted (at least
         sorted for the first letter).
         Note that this table actually points into the RexxList
         table containing all Rexx commands (see 'pvcall 39').
         Note that <type> is not used by PowerVisor but is used by
         the ARexx interpreter
            -> <pointer to routine table>
                  <pointer to string>.L <type>.L <pointer to routine>.L
                  .
                  .
                  .
                  0.L 0.L

38       Mode masks and values. You may change this table. This list
         need not be sorted
            -> <pointer to mode routine table>
                  <pointer to string>.L <mask>.L <value>.L
                  .
                  .
                  .
                  0.L

39       RexxList. You may change this table. The first part of this
         table consists of all PowerVisor functions. The second part
         of table (also pointed to by 'pvcall 37') consists of all
         PowerVisor commands.
            -> <pointer to rexx command list>
                  <pointer to string>.L <type>.L <pointer to routine>.L
                  .
                  .
                  .
                  0.L 0.L 0.L
               <type>
                  0 = Normal function, returns number
                  1 = String function, returns string

40       ScreenBase
            -> <ScreenBase>

41       MemoryBase
            -> <MemoryBase>
```

```
42       OBSOLETE

43       Get pointer to stringgadget
            -> <stringgadget>
               Fields in the StringGadget structure.
                  offs  size  function
                  ---------------------------------
                  0     4     NextGadget
                  4     2     LeftEdge
                  6     2     TopEdge
                  8     2     Width
                  10    2     Height
                  12    2     Flags
                  14    2     Activation
                  16    2     GadgetType
                  18    4     GadgetRender
                  22    4     SelectRender
                  26    4     GadgetText
                  30    4     MutualExclude
                  34    4     SpecialInfo
                  38    2     GadgetID (not used by PowerVisor)
                  40    4     UserData (not used by PowerVisor)

44       OBSOLETE

45       OBSOLETE

46*      Error handler. The error handler executes a routine (pointer
         in a5). If there is any error in the routine, control will
         return back to after the call of this routine (the 'Z' flag
         will be set to indicate that there was an error). All registers
         are preserved for the routine.
            <a5 = pointer to routine>

47       Install a command that will be executed before PowerVisor
         quits. Using this function you can cleanup your memory
         before it is too late. If you return 0 from this function
         the quit will not happen.
            <commandstring>

48       Search the alias list and return the converted command. If the
         command is not in the alias list the original commandline is
         returned. Note that this function always returns a pointer to
         a new string. You must free this string later with 'pvcall 51'.
            <string>
            -> <new string> (PV block)

49*      Add a memory region allocated with 'pvcall 50' to the
         autoclear list. The autoclear list contains at most 10
         (by default, you can change this value in 'MemoryBase')
         allocations. If more than 10 allocations are added the
         last allocation (timewise) is removed and freed. This means
         that this method is not absolutely safe, but safe enough
         for most purposes. The autoclear list is mostly used for
         strings.
         Note that it is not possible to remove something from this
```

list. This means that once some pointer is added you may never free the pointer yourselves.
PowerVisor also frees all memory in this list before quiting. Note that PowerVisor uses this list for all strings and string pointers the user uses.
<d0 = pointer>
-> Z flag is set if there was an error

50* Allocate a block of memory. The memoryblock allocated with this function is called a PV block (do not confuse with PV memoryblock since this is something completely different). A PV block is a pointer after the size. This size is contained in a word if the block is smaller than a 65533 bytes. Else it is containted in a longword.
Note that you must explicitely free this block with 'pvcall 51' except if you add this block to the autoclear list with 'pvcall 49' or to the global autoclear list with 'pvcall 55'.
<d0 = size>
-> <d0/Z flag = pointer to PV block or 0 if error>

51* Free a PV block. Do not free a PV block when it is added to the autoclear list using 'pvcall 49' or when it is added to the global autoclear list with 'pvcall 55'. Generally it is not safe to free memory not allocated with 'pvcall 50' (there are exceptions like 'pvcall 48' for example).
<a0 = pointer>

52 Compute the gadget and the intuition signal bits. Use this function when you have changed something to the StringInfo structure or the Gadget structure, or when you have changed the IDCMP values for the PowerVisor window.

53* Print a string. The printing will stop when the 0 character is encountered in the string or when d3 characters are printed. You may also enclose linefeed characters in the string (ascii 10).
<a0 = pointer to string>
<d3 = length>

54* Print a number. Note that this function may be interrupted by the user. If you want to be absolutely sure you should use the errorhandler ('pvcall 46') for this routine.
Note that the previous routine ('pvcall 53') is safe and can't be interrupted.
<d0 = number>

55* Add a pointer to a PV block to the global autoclear list. This is the list where all allocations from the 'alloc' function reside. Note that when you have added the pointer to this list you must not forget to remove the pointer from the list when you free the PV block with 'pvcall 51' (Use 'pvcall 56' for this purpose). (Use the showalloc command to see all allocations in this list).
<d0 = pointer to PV block>
-> <Z flag is true if not enough memory to add it>

```
56*     Remove a pointer to a PV block from the global autoclear list.
        Note that you are still responsible for freeing the PV block.
           <a0 = pointer to PV block>

57*     Close a PV handle.
        You must remember that when you close a standard PV handle
        (like the PV handle for the help file), you MUST set the handle
        value in the corresponding base to 0. Otherwise PowerVisor will
        try to close the file again.
           <d1 = pointer to PV handle>

58*     Reallocate a PV memory block.
           <a0 = pointer to PV memory block>
           <d0 = new size (if 0 block is freed)>
           -> <a0 = pointer to the same PV memory block (unchanged)>
           -> <d0/Z flag = pointer to memory or 0 if no success>

59      OBSOLETE (Since V1.32)

60      OBSOLETE

61*     Refresh a logical window
           <a0 = pointer to logical window>

62*     Snap a word from a position in a logical window
           <a0 = pointer to logical window>
           <d0 = x position (relative to physical window)>
           <d1 = y position>
           <a1 = buffer for word>
           <d2 = length of buffer>
           -> <d0/Z flag = resulting length of buffer or 0>

63*     Disassemble some memory
           <a0 = pointer to string space (make it big enough)>
           <d0 = address to disassemble>
           <a6 = pointer to library (or NULL), this is in fact the
                 contents of the 'a6' variable. When a6 <> 0 PowerVisor
                 will disassemble library calls with the correct
                 names instead of the offset>
           -> <d0 = number of bytes disassembled>
           -> <a0 = pointer to end of string>

64      Disassemble some memory. Commandline version
           <string pointer>
           <address>
           <library pointer or 0>
           -> <number of bytes disassembled>

65      OBSOLETE

66*     Allocate some memory in the internal PowerVisor memory pool.
        All memory allocated here is automatically freed when
        PowerVisor quits. This function represents the lowest level
        of allocation routines in PowerVisor. Normally you do not
        need this function. Use the higher level functions instead
        (see above)
           <d0 = size in bytes>
```

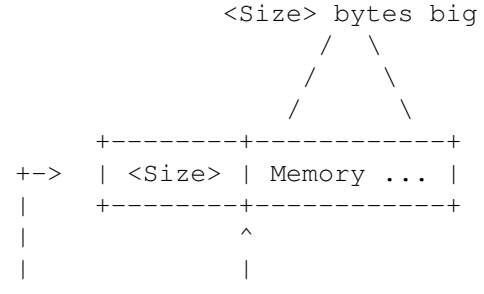
```
            <d1 = attributes like MEMF_CHIP and MEMF_CLEAR)>
            -> <d0/Z flag = pointer to allocated memory or 0>

67*         Free some memory allocated with 'pvcall 66'. This function
            represents the lowest level of allocation routines in
            PowerVisor. Normally you do not need this function. Use the
            higher level functions instead (see above)
               <a1 = pointer to memory allocated with 'pvcall 66'>
               <d0 = size in bytes>

68*         Reallocate some memory in the internal PowerVisor memory pool.
            This is memory allocated with 'pvcall 66'. This function
            represents the lowest level of allocation routines in
            PowerVisor. Normally you do not need this function. Use the
            higher level functions instead (see above)
               <a1 = pointer to memory allocated with 'pvcall 66'>
               <d0 = old size in bytes>
               <d1 = new size in bytes>
               <d2 = attributes (MEMF_CHIP) for the new block. These
                     attributes are only used when the block must be
                     moved because there is not enough room to grow
                     a block>
               -> <d0/Z flag = pointer to new block> Note that when the
                     allocation failes, the old block is NOT freed
```

1.4 The Wizard Corner : Description of internal memory formats

Almost all memory in PowerVisor is allocated in the internal memory management system. Normally you don't need to concern yourselves with that system since the higher level routines take care of it. So you should use 'alloc()', 'free()', 'realloc()', 'pvcall 49', 'pvcall 50', 'pvcall 51', 'pvcall 55', 'pvcall 56' and 'pvcall 58' to handle memory instead of 'pvcall 66', 'pvcall 67' and 'pvcall 68'. Except maybe in some rare cases.

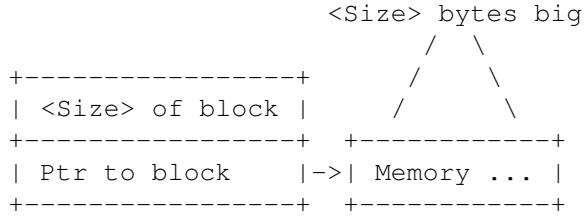
Here are all the highlevel memory allocation elements used by PowerVisor :

PV block
: A PV block is a pointer to memory. It is used very often. You can use the 'pvcall 50' and 'pvcall 51' functions to allocate or free such blocks. Be careful when you free PV blocks that you have not allocated. If you want to be totally safe you should always clear the variable in the appropriate base when you free a PV block. If it is absolutely unsafe to free a certain PV block, a warning will be given in the description (see below). Otherwise you may assume that you can use the PV block.



```
                  |       PV block points to this
                  |
                  +------ Size is 2 bytes or 4 bytes long
                          If size is 2 bytes long the PV block will not be
                          long word alligned.
```

PV memory block
   A PV memory block is a relocatable piece of memory. You can use
   'pvcall 58' to manage this memory. Note that after a reallocmem
   ('pvcall 58') the memory block can be moved to another place.



PV handle
   PV handles are filehandles used by PowerVisor for buffered file IO.
   The only operation you can do on a PV handle is 'pvcall 57'.

EXEC block
   A normal block allocated with AllocMem ('pvcall 66') (NOT with the
   Exec AllocMem).

DOS file
   A normal DOS filehandle (BPTR).

1.5 The Wizard Corner : The bases

The rest of this chapter is dedicated to the internal data structures. You
can get the pointers to these structures with the 'pvcall' commands.
Read-only fields are indicated with an asterix ('*') in front of the line.
A read-only field does not always mean that changing it may harm
PowerVisor. It can also indicate that PowerVisor only uses the field once
and changing it won't have any effect.
All other fields can be modified but you must make sure that you follow
the conventions: A PV block must remain a PV block and so on.
You can use 'pvcall 51' to free a PV block. When there is some restriction
on the use of an internal variable it is mentioned in the list.

Note that all structures described below are also available in include
file form (both .h and .i include files). See the 'PVDevelop' subdirectory.

1.6 The Wizard Corner : main base (pvcall 36)

	Offset	Size	Function
*	0	2	OS version (1 if 2.0 or higher)
*	2	4	DosBase

```
*  6          4         IntuitionBase
*  10         4         GraphicsBase
*  14         4         UtilityBase (0 in AmigaDOS 1.2/1.3 version)
*  18         4         ExpansionBase
*  22         4         DiskFontBase
*  26         4         PowerVisorBase
*  30         4         CLI commandline for PowerVisor
   34         4         If TRUE PowerVisor is detached
   38         4         Pointer to error file handle (DOS file)
<  42         8         >
   50         2         Speed of refresh
   52         2         Refresh counter
   54         4         Command that is refreshing (PV block)
   58         (2+2)*6   Codes (WORD) /Qualifier (WORD) table

                          Key      Default code   Default Qualifier
                          ------------------------------------------
                          Break    ESC            none
                          HotKey   /              right-shift+right-alt
                          Pause    HELP           right-alt
                          NextWin  TAB            none
                          HistUp   UPKEY          none
                          HistDo   DOWNKEY        none

                         Same keys as in 'prefs key' command
   82         1         If equal to 1 we are in PowerVisor debug mode
                        (see 'pvcall 17')
<  83         3         >
   86         4         'Pre' command (PV block)
   90         4         'Post' command (PV block)
   94         4         'Quit' command (PV block)
   98         4         Pointer to last history string in history buffer
                        (or 0 if the history buffer is empty). This is
                        the first history line that is going to be deleted
                        when there are too many lines in the history
                        buffer. For the format of history lines see below
   102        2         Last error code
   104        2         Execution level (pvcall 18)
*  106        (4+4)*6   Signal bitnumbers (LONG) and signal sets (LONG)

                           Hold
                             Sending this singal to PowerVisor will cause
                             PowerVisor to reopen it's screen after a
                             'hold'.
                           PortPrint
                             Use this signal bit in conjunction with
                             sending a message on the PowerVisor port.
                           IDC
                             (Input Device Command) Using this signal you
                             can execute IDC commands.
                           GadgetRefresh (to PowerVisor.task)
                             Say to PowerVisor.task that the stringgadget
                             needs refreshing.
                           PVtoFront (to PowerVisor.task)
                             Say to PowerVisor.task that PowerVisor should
                             come to the front. Sending this signal also
                             causes a 'Hold' signal to PowerVisor.
```

```
                          InterruptPV
                            Interrupt PowerVisor.

*  154          4          PowerVisor.task
*  158          4          Input request block
*  162          4          Input device port
   166          4          Pointer to first history line

                           The format of one history line is the following :
                              <next>.L <prev>.L <Size>.W <string>
                           One history line is a simple EXEC block. If you
                           want to free one you must make sure that you
                           use <Size>.W for size, and that the double linked
                           list remains correct, 166:4 (this field) must
                           point to the first history line (may be 0 if
                           there are no history lines) and 98:4 must point
                           to the last history line in the history buffer.
                           Note that <prev>.L is 0 for the first history
                           line (the one pointed to by 166:4) and <next>.L
                           is 0 for the last history line (the one pointed
                           to by 98:4).
                           Also make sure that 252:4 (the pointer to the
                           history line we are scanning) points to 0 (the
                           easy way) or points to an existing history line
                           (the hard way) when you delete a line.
                           If you delete or add a history line you must also
                           make sure to update 170:4 (this is not neccessary
                           if you use the standard pvcall functions to add
                           a history line)

   170          4          Number of lines in history
   174          4          Maximum number of lines in history (default 20)
   178          32         Code table
                              Each bit in this table represents a code. If
                              the bit is 1 this means that there is a macro
                              with this code defined.
   210          4          Pointer to first alias structure (or 0 if there
                           are no aliases)
                             Each alias structure looks as follows :

                             offs  size  function
                             ------------------------------------------------
                             0     4     Next alias string (0 for last)
                             4     4     Previous alias string (0 for first
                             8     4     Pointer to command string (PV block)
                             12    4     Pointer to alias string (PV block)

                             It is safe to change this list and to replace
                             strings as long as you respect the double linked
                             list and give valid PV blocks in each structure.
                             You may free the two strings (with pvcall 51)
                             if you replace them with other PV blocks.
                             Note that an alias structure is an EXEC block.

   214          4          Pointer to scriptline
                           Make sure that you respect the maximum line length
                           (see 218:2) when you change this pointer.
```

```
   218         2         Default line length
   220         1         Character used for comments (default ;)
   221         1         Character used for feedback suppress (default ~)
   222         1         Character used for quick exec (default \)
   223         1         Character used to suppress output (default -)
   224         1         Last command
                            0 normal command
                            1 memory command
                            2 unasm
                            3 view
<  225         1         >
<  226         1         >
*  227         1         If 1 we are in hold mode (screens are closed)
<  228         1         >
   229         1         Input device command number

                          nr    name        function
                          ------------------------------------------------
                          1     NEXTWIN     Make next logwin the scroll
                                            window
                          2     SCROLL1UP   Scroll logwin one line up
                          3     SCROLLPGUP  Scroll logwin five lines up
                          4     SCROLLHOME  Scroll to home position
                          5     SCROLLEND   Scroll to bottom position
                          6     SCROLL1DO   Scroll one line down
                          7     SCROLLPGDO  Scroll five lines down
                          8     SCROLLRIGHT Scroll to the complete right
                                            side
                          9     SCROLL1RI   Scroll one column right
                          10    SCROLL1LE   Scroll one column left
                          11    DSCROLL1UP  Scroll debug window one word
                                            up
                          12    DSCROLLPGUP Scroll debug window 20 words
                                            up
                          13    DSCROLL1DO  Scroll debug window one word
                                            down
                          14    DSCROLLPGDO Scroll debug window 20 words
                                            down
                          15    DSCROLLPC   Scroll debug window to PC
                          16    EXEC        Execute command (ptr in 230:4)
                          17    SNAP        Snap string (ptr in 230:4)
                          18    DSCROLL1IUP Scroll one instruction up
                          19    DSCROLL1IDO Scroll one instruction down

   230         4         Pointer to argument for IDC command EXEC.
   234         14        List containing the macros (key attachements).
                         One macro node looks like this :

                          offs  size  function
                          ------------------------------------------------
                          0     4     ln_Succ
                          4     4     ln_Pred
                          8     1     ln_Type
                          9     1     ln_Pri (not used)
                          10    4     ln_Name (not used)
                          14    2     Code for key
                          16    2     Qualifier
```

```
                              18    4     Commandstring (EXEC block)
                              22    2     Length of command string
                              24    2     Flags
                                             INVISIBLE = 1  If set, command is
                                                not added to stringgadget
                                                before it is executed. It is
                                                executed with IDC commands
                                             SNAP = 2. If set, command is
                                                snapped to the current
                                                position in the stringgadget.
                                                Nothing is executed
                                             HOLDKEY = 4. If set, the attached
                                                key is not removed from the
                                                input event list

   248          4          Pointer to workbench message (or 0 if started
                           from cli)
   252          4          Pointer to history line we are scanning, if 0
                           we are typing a new line or the stringgadget
                           is empty (See the history variables above for
                           more information).
                           This pointer is used by the general input routine
                           and by the input handler to scan through the
                           history buffer. You can use pvcall 2 and pvcall 3
                           to change this pointer or you can change it
                           yourselves
   256          22         Input event used by the 'AddEvent' command
*  278          4          LayersBase
<  282          4          >
*  286          1          If true this is the master PowerVisor
```

1.7 The Wizard Corner : eval base (pvcall 30)

```
   Offset       Size       Function
-------------------------------------------------------------------------------
<  0            8          >
   8            8          Variables and functions (PV memory block)
                              Format for variables and functions:
                                 <Value or pointer>.L <Name len>.B <Type>.B
                                 <Name> [<pad>.B] [<spec>.L]
                              <Type>
                                 0 = variable
                                 1 = constant
                                 2 = special
                                 3 = function
                              <spec> is pointer to routine to call when
                                 variable changes (only when <Type> == 2)
                           Note that you better not change the variables
                           'error' and 'rc'. These should remain on the
                           same position. This is because PowerVisor accesses
                           these variables with a fixed offset from the start
                           of the variable list.
   16           18         Operator priorities
                           One byte for each operator. Priorities between
                           1 and 10 are supported (1 is low priority)
```

Op	Function	Default priority
^	Xor	4
&	And	5
|	Or	3
*	Multiply	10
/	Divide	10
%	Modulo	10
+	Add	9
-	Subtract	9
>	Greater than	7
<	Less than	7
>=	Greater or equal	7
<=	Less or equal	7
!=	Not equal	6
==	Equal	6
<<	Left shift	8
>>	Right shift	8
&&	Logical and	2
||	Logical or	1

1.8 The Wizard Corner : arexx base (pvcall 31)

	Offset	Size	Function
*	0	4	Rexx signal bit
	4	2	Sync flag (if 1 we are in Sync)
*	6	2	Hide flag (if 1 we are in Hide)

1.9 The Wizard Corner : debug base (pvcall 32)

	Offset	Size	Function
*	0	4	If floatingpoint coprocessor present this variable contains 4, else 0
	4	14	List containing all debug tasks. One debug node looks like this :

offs	size	function
0	4	ln_Succ
4	4	ln_Pred
8	1	ln_Type
9	1	ln_Pri (not used)
10	4	ln_Name
14	4	MatchWord = 'DBUG'
18	1	Mode (mode)

Nr	Name	Function
0	NONE	Doing nothing

		1 TRACE	Tracing
		2 EXEC	Executing
		3 FLOWT	Flow tracing (68020)
19	1	SMode (special mode)	
		Nr Name	Function
		0 NORMAL	Normal debugging
		1 TTRACE	Temporary trace
		2 CRASH	There was a crash
		3 BREAK	There was a breakpoint
		4 TBREAK	Break due to trace
		5 WAIT	Waiting for PowerVisor
		6 ERROR	There was an error
20	4	BPTR to loaded segment (only with 'debug l')	
24	4	Address of instruction to execute	
28	4	Pointer to temporary routine	
32	4	Pointer to trace exception routine	
36	4	Address to restore breakpoint (only if SMode = TTRACE)	
40	4	Additional information for tracing.	
44	1	TMode (trace mode) <n> is 40:4	
		Nr Name	Trace Function
		0 NORMAL	Normal
		1 AFTER	<n> instructions
		2 STEP	endlessly
		3 UNTIL	until pc=<n>
		4 REG	until register changes (OBSOLETE)
		5 COND	until condition true
		6 BRANCH	until branch
		7 FORCE	force tracing (trace f)
		8 OSCALL	until OS call used
		9 SKIP	for trace t
		10 QCOND	quick condition
		11 PROF	profile tracing
<45	1	>	
46	1	TDNestCnt	
47	1	IDNestCnt	
48	1	TaskState (TS_READY or TS_WAIT)	
49	1	Dirty. If 1 our debug window needs full refreshing. If 2 our source window needs full refreshing. If 3 both windows need refreshing	
50	4	TC_SIGWAIT	
54	4	Crash number	
58	4	Additional argument for some trace	

```
              modes
  62    4     Pointer to task corresponding with
              debug node
  66    4     Top PC visible in debug window
  70    4     Bottom PC visible in debug window
 <74    2     >
  76    32    Number of bytes for each instruction
              on screen (32 bytes, one byte for
              each line)
  108   4     Initial programcounter
  112   4     Previous trapcode for task
  116   8     PV memory block for symbol values.
              Each element in this block is a
              value and an offset in the following
              string quickblock (8 bytes per
              entry)
  124   4     Pointer to the source structure
              where the program counter is
              located
  128   8     PV memory block containing all
              strings for the symbols. All strings
              in this block are null terminated.
  136   4     Linenumber in source where the
              program counter is located
  140   14    Breakpoint list.
 <154   2     >
 <156   4     >
 <160   2     >
 <162   1     >
 <163   1     >
  164   4     Pointer to quit code on stack.
  168   4     Original quit code. (Code that is
              called when the task quits).
  172   4     SP
  176   4     PC
  180   2     SR
  182   15*4  Registers
 <242   16    >
  258   4     Pointer to first source structure
  262   4     Pointer to current source structure
  266   2     If 1, logical window is locked
              (does not follow the program
              counter) (like 'source h')
```

One breakpoint node looks like this :

```
  offs  size  function
  ------------------------------------------------
  0     4     ln_Succ
  4     4     ln_Pred
  8     1     ln_Type
  9     1     ln_Pri (not used)
  10    4     ln_Name (not used)
  14    2     Number
  16    4     Address of breakpoint
  20    2     Original contents of memory
  22    1     Type
```

```
                                  T  temporary breakpoint
                                  t  temporary breakpoint
                                     (internal)
                                  N  normal breakpoint
                                  P  profile breakpoint
                                  C  conditional breakpoint
                                  A  break after <n> passes
                                  s  temporary breakpoint
                                     (internal)
                          <23     1     >
                           24     4     Usage count
                           28     4     Additional argument
                                        conditional string if type is 'C'
                                        breaknumber if type is 'A'
                           32     4     Routine to jump to if a break occurs
                          <36     4     >

                         One source structure looks like this :

                           offs  size  function
                           ------------------------------------------------
                           0     4     Next source structure
                           4     4     Previous source structure
                           8     4     Pointer to source filename
                           12    4     Size of following block
                           16    4     Pointer to the block with linenumber
                                       information. Each info block is 8
                                       bytes long. The first long is
                                       the address and the second long is
                                       the line number in this source
                           20    4     Size of the following block
                           24    4     Pointer to the loaded file or 0
                           28    4     Current linenumber for program-
                                       counter
                           32    4     Top linenumber visible in Source
                                       logical window
                           36    4     Bottom linenumber visible in Source
                                       logical window
                           40    4     Current hilighted linenumber
                                       (linenumber in Source logical
                                       window, not the linenumber in the
                                       file)

    18          4         Current debug task
<   22          16        >
<   38          8         >
    46          1         Show register info after each trace (default 1)
    47          1         Give disassembly after each trace (default 1)
    48          2         Number of lines to disassemble (default 5)
    50          2         Show previous instruction after each trace
                          (default 1)
    52          2         Tab size for source level debugger (default 8)
```

1.10 The Wizard Corner : file base (pvcall 33)

	Offset	Size	Function
	0	4	Pointer to CLI outputhandle (DOS file)
	4	4	Pointer to control file (PV handle)
	8	4	Pointer to help file (PV handle)
	12	4	Pointer to script file (PV handle)
	16	4	Pointer to log file (DOS file)
	20	4	Pointer to log logical window

1.11 The Wizard Corner : general base (pvcall 34)

	Offset	Size	Function
*	0	4	Pointer to PowerVisor (process)
	4	4	Lower bound for stack pointer when PowerVisor should give a 'Possible stack overflow' error. This pointer is 512 bytes away from the TC_SPLOWER value of the PowerVisor task. You can change this value if you think it is not safe enough or it is too safe. This bound is checked whenever a command is executed (a group is not a command but a group of commands) and in the recursive part of the expression evaluator.
*	8	4	Trackdisk request block
*	12	4	Trackdisk port
*	16	4	Old ExecTrapCode
*	20	4	MMUType 0 = no MMU 68851, 68030 or 68040
*	24	2	1 if 68020 or higher, else 0
*	26	4	Block with account tasks
*	30	4	Old Switch function
*	34	4	Old Alert function
*	38	4	Old AddTask function
*	42	4	Old AutoRequest function
	46	4	Stack fail level (default 40)
<	50	8	>
	58	14	List with freezed tasks
	72	14	List with crashed tasks One crash node looks like this :

offs	size	function
0	4	ln_Succ
4	4	ln_Pred
8	1	ln_Type
9	1	ln_Pri (not used)
10	4	ln_Name
14	4	Crashed task
18	4	TrapNumber
22	4	2ndInfo (from Alert)

```
                    26    1     0 if trap, 1 if guru, 2 if stack
                                fail
                   <27    1     >
                    28    4     SP
                    32    4     PC
                    36    2     SR
                    38    15*4  Registers

86          14      List with fd-files
                    One fd-file node looks like this :

                    offs  size  function
                    ------------------------------------------------
                    0     4     ln_Succ
                    4     4     ln_Pred
                    8     1     ln_Type
                    9     1     ln_Pri (not used)
                    10    4     ln_Name
                    14    4     Library
                    18    2     Bias
                    20    8     PV memory block containing all
                                functions
                    28    8     PV memory block containing all
                                strings
                    36    2     Number of functions

100         14      List with functions we are monitoring (see the
                    'AddFunc' command)
                    One function node looks like this :

                    offs  size  function
                    ------------------------------------------------
                    0     4     ln_Succ
                    4     4     ln_Pred
                    8     1     ln_Type
                    9     1     ln_Pri (not used)
                    10    4     ln_Name
                    14    4     Library
                    18    2     Offset
                    20    4     Task to monitor (if zero, all tasks)
                    24    4     Usage count
                    28    4     Pointer to count code (EXEC block)
                    32    4     Size of count code
                    36    4     Old function to restore later
                    40    2     Type flags

                                   0 = Normal
                                   1 = Led
                                   2 = With register information
                                   3 = Led and register information
                                   8 = Exec

                    42    2     Position in following block where
                                the last added task is added
                    44    8*4   8 pointers to the 8 last tasks using
                                this function
                    76    8*56  All registers for each task (d0-d7/
```

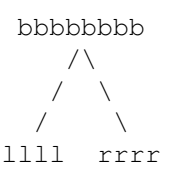
```
                                    a0-a5)
                          524   4     Ptr to command

   114          34+6      The PowerVisor port. This is an Exec message port
                          followed by a longword containing the pointer to
                          the PVCallTable and a private word.
                          You can find the pointer to the PVCallTable at
                          offset 34 in this port.
<  154          1         >
   155          1         Old priority (before PowerVisor set it to 4)
*  156          4         Timer device request block (for 'stack' command)
*  160          4         Timer device port
   164          4         Maximum stack usage (like 'getstack' function)
   168          4         Task we are looking at with 'stack' command
   172          4         Number of microseconds to wait
   176          4         Task we are tracking (with the 'track' command)
   180          4         Pointer to the first element in the double linked
                          track list. There is one element in this list
                          for each 'AllocMem', 'AllocVec' or 'OpenLibrary'.
                          One track structure looks like this :

                            offs  size  function
                            ------------------------------------------------
                            0     4     Next track structure
                            4     4     Previous track structure
                            8     4     Pointer to tracked data (result
                                        from 'AllocMem' or 'AllocVec' or
                                        pointer to library for
                                        'OpenLibrary')
                            12    4     Size of allocation for 'AllocMem'
                                        or 'AllocVec' or version of
                                        library for 'OpenLibrary'
                            16    4     Program counter
                            20    1     Type
                                           0 = AllocMem
                                           1 = OpenLibrary
                                           2 = AllocVec
                                           3 = AllocSignal
                                           4 = CreateMsgPort
                                           5 = CreateIORequest
                                           6 = Lock
                                           7 = Open
                                           8 = AllocRaster
                             <21    3     >
   184          4         Pointer to debug node we are profiling
   188          4         Number of micro seconds to wait
   192          4         Number of ticks the debug node was in
                          'WAIT' state
   196          4         Number of ticks the debug node was in
                          'READY' state
   200          8         PV memory block to profiler table
```

1.12 The Wizard Corner : list base (pvcall 35)

	Offset	Size	Function
*	0	4	Old WindowPtr from PowerVisor process
	4	4	Prompt string
	8	2	Current list number
<	10	2	>
	12	14	List containing all structure nodes. (Warning ! Structure nodes and structure definitions are not the same) One structure node looks like this : offs size function 0 4 ln_Succ 4 4 ln_Pred 8 1 ln_Type 9 1 ln_Pri (used to sort the nodes by length of name) 10 4 ln_Name 14 4 MatchWord = 'PVSD' 18 4 Pointer to string block (PV block) 22 4 Pointer to structure definition (PV block), (see below) 26 2 Length of structure Structure definitions look like this : { <String>.L <Type>.W <Offset>.W ... } 0.L 0.L <String> is a pointer to the string corresponding with the name of a structure element. <Type>.W can be somethine like : 0 = byte 1 = word 2 = long 3 = string 4 = object in object (like ViewPort in Screen) To do BPTR to APTR conversion you must add 128 to this word. <Offset>.W is the offset of the element in the structure.
	26	30*40	All infoblocks (see above) for all standard lists. In the following order (size is 28*40 for AmigaDOS 1.3) : Exec Intb Task Libs

```
  Devs
  Reso
  Memr
  Intr
  Port
  Wins
  Scrs
  Font
  Dosd
  Func
  Sema
  Resm
  Fils
  Lock
  IHan
  FDFi
  Attc
  Crsh
  Graf
  Dbug
  Stru
  PubS (not in AmigaDOS 1.3 version)
  Moni (not in AmigaDOS 1.3 version)
  Conf
  LWin
  PWin

An infoblock is a description of a list.
One infoblock looks like this :

  offs  size  function
  ------------------------------------------------
  0     4     Prompt string
  4     1     Item number
  5     1     Control byte. This byte controls how
              you should go to the start of
              the list.

                 -1  = routine (like 'DosD')
                       6:4 is pointer to routine
                       to call to go to the first
                       element of the list
                 -2  = structure (like 'Exec')
                       6:4 is pointer to pointer
                       to structure
                 -3  = (like 'Fils')
                       6:4 is pointer to routine
                       to call. This routine will
                       do the complete list
                       without any intervention at
                       all

                 If Control is not equal to -1, -2
                 or -3 the start of the list is
                 computed as follows :

                    The byte is split in two
```

nibbles :



The value in 6:4 is loaded. If 'llll' is equal to 0 nothing happens with this value, if it is equal to 1 you must take the indirection one step further (take the contents of the value), if it is equal to 2 you must first convert the value from BPTR to APTR before you take the contents of this value.

We continue with the value obtained from the previous algorithm and add 10:2 to it. Now we look at 'rrrr'. If it is equal to 0 we do nothing, if it is equal to 1 we take the contents of this value, if it is equal to 2 we convert the value from BPTR to APTR before we take the contents.

Now we have computed the address of the first element in the list.

6	4	Pointer to the routine to go to the base of a list or the pointer to the base of the list (what it really is depends on the value of the control byte 5:1) The routine must return the pointer to the first list element in a2. This routine may initialize d7 for use by the next element routine (see below).
10	2	Offset to add to 6:4 (depending on the control byte 5:1)
12	4	If control byte is -2 this variable contains the pointer to the structure definition (not the node) (structure definitions are described above) If control byte is -3 this variable is not used. Else this variable contains the pointer to the routine to go to the next element in the list. This routine must preserve a0 and a1. d7 is free to be used as an external

```
                                variable (may be setup by startup
                                function). a2 is pointer to list
                                element currently listing. This
                                routine must return the pointer
                                to the next element in the list in
                                a2 and set the Z flag to true if the
                                end of the list is reached.
                  16    4       Pointer to header string
                  20    4       Pointer to format string (RawDoFMT
                                format)
                  24    4       Argument string for 'list' command
                  28    1       Must contain 0
                  29    1       If true, 30:4 is a pointer to a
                                structure definition (not the node)
                                containing all the information to be
                                printed when the 'info' command is
                                used.
                                Else 30:4 is a pointer to a routine
                                doing the same thing.
                  30    4       Routine or structure definition. for
                                the 'info' command. If 0, there is
                                no more info for this list.
                  34    4       Pointer to routine printing one line
                                for one element of the list.
                                This routine expects the pointer to
                                the list element in a2.
                  38    2       Offset for the name element in
                                the structure
```

1.13 The Wizard Corner : screen base (pvcall 40)

```
   Offset      Size      Function
-----------------------------------------------------------------------------
<  0           1         >
<  1           1         >
<  2           1         >
<  3           1         >
<  4           1         >
   5           1         If true PowerVisor will not clear line the next
                         time 'Scan' is started.
<  6           1         >
<  7           1         >
<  8           1         >
<  9           1         >
*  10          4         Length of stringgadget buffer line (not used)
   14          4         Pointer to stringgadget buffer
   18          2         Position of cursor in stringgadget
   20          (2*4)*7   Default sizes and parameters for each logical
                         window (like 'prefs logwin' command)
                             <columns>.W <rows>.W <mask>.W <flags>.W
                             .
                             .      (6 times for each logical window)
                             .      (Main,Extra,Refresh,Debug,PPrint,
                             .       Rexx,Source)
                             .
```

```
   76          4         Pointer to 'snap' command (PV block)
<  80          1         >
<  81          3         >
<  84          4         >
   88          4         Pointer to 'Main' physical window
                         A physical window structure looks like this :

                           Offs  Size  Function
                           ---------------------------------------------
                           0     4     ln_Succ
                           4     4     ln_Pred
                           8     1     ln_Type
                           9     1     ln_Pri (not used)
                           10    4     ln_Name (EXEC block)
                           14    48    NewWindow structure
                           62    4     Window
                          <66    4     >
                           70    2     Last code for VANILLAKEY
                           72    2     Last qualifier
                           74    1     LeftBorder for masterbox
                           75    1     TopBorder
                           76    1     RightBorder
                           77    1     BottomBorder
                           78    4     Pointer to masterbox
                           82    4     Pointer to Global structure
                           86    14    Logical window list
                          <100   4     >
                           104   4     Gadget list (AmigaDOS 2.0 only)

   92          4         Pointer to 'Main' logical window
                         A logical window structure looks like this :

                           Offs  Size  Function
                           ---------------------------------------------
                           0     4     ln_Succ
                           4     4     ln_Pred
                           8     1     ln_Type
                           9     1     ln_Pri (not used)
                           10    4     ln_Name (EXEC block)
                           14    4     pointer to box
                           18    2     x real coordinate (in physical
                                       window)
                           20    2     y real coordinate
                           22    2     w real width
                           24    2     h real height
                           26    2     first visible column in logical
                                       window
                           28    2     first visible row
                           30    2     current column coordinate
                           32    2     current row coordinate
                           34    2     visible width in characters
                           36    2     visible height in characters
                           38    4     flags

                                          1     Print on file
                                          2     Print on screen
                                          4     Enable -MORE- check
```

```
                  8     private
                  16    private
                  32    Total home is equal to
                        (0,0)
                  64    statusline on/off
                  128   breakcheck on/off
                  256   auto output snap
                  512   (if false) add scrollbar
                        to logical window if 'mode
                        sbar' is set
                  1024  (if true) always add
                        scrollbar (ignore previous
                        flag)

 42    8      TextAttr
              A TextAttr structure looks like
              this :

                  Offs  Size  Function
                  ------------------------------
                  0     4     Name
                  4     2     YSize
                  6     1     Style
                  7     1     Flags

 50    4      Pointer to font
 54    2      Font X character width
 56    2      Font Y character height
 58    2      Font baseline
 60    4      Pointer to physical window (MainPW)
 64    2      number of columns optimal
 66    2      number of rows optimal
 68    2      number of lines in buffer
 70    2      number of columns per line in buffer
 72    4      pointer to buffer
              These is a table of (68:2)+1
              pointers to lines. Each line is
              70:2 chars long (plus one for the
              attribute in the beginning of the
              line) If this attribute is non-null
              the line will be hilighted:
                  1 = hilighted
                  2 = inverse video
                  3 = hilighted and inverse video
 76    4      Log file (DOS file)
 80    2      number of lines passed (for -MORE-
              check)
<82    4      >
 86    4      Pointer to extra title (user of
              logical window is responsible for
              remembering and freeing the memory
              for this title)
 90    1      if TRUE we are active
 91    1      TopBorder used for statusline (10 if
              statusline or 0 if no statusline)
 92    2      real top coordinate
              92:2 = 20:2-91:1
```

Offs	Size	Function
94	4	userdata (used by PowerVisor)
98	2	The number of the hilighted line in the logical window (or -1 if no line is hilighted)

A box structure looks like this :

Offs	Size	Function
0	4	Parent box or NULL if masterbox
4	4	Child A (not used if box is ATOMIC)
8	4	Child B
12	4	Logical window (only if box is ATOMIC)
16	4	Physical window
20	2	Share for child A (in % x 10)
22	1	Type 0 UPDOWN 1 LEFTRIGHT 2 ATOMIC
23	1	If true our box needs a cleanup
24	1	Left border for inner box
25	1	Top border
26	1	Right border
27	1	Bottom border
28	2	x position after accounting for window and inner box. These variables define the box that we really can use for output (for the logwin)
30	2	y
32	2	w
34	2	h
36	2	x1 position for titlebar
38	2	y1
40	2	x2
42	2	y2
44	4	pointer to gadget (AmigaDOS 2.0) for scrollbar (GadTools)
48	4	pointer to NewGadget structure

96	4	Pointer to 'Refresh' logical window
100	4	Pointer to 'Debug' logical window
104	4	Pointer to 'Extra' logical window
108	4	Pointer to 'PPrint' logical window
112	4	Pointer to 'Rexx' logical window
116	4	Pointer to 'Source' logical window
120	4	Pointer to current logical window
124	2	location of horizontal prompt (default 1) relative to left side of window (plus border).
126	2	left location of stringgadget relative to left side of window (plus border) (default 50).
128	2	offset for right side of stringgadget relative to the rightside of the window (default 0).
130	4	Pointer to PowerVisor steal screen This is 0 if PowerVisor is on its own screen
134	4	Pointer to PowerVisor real screen

```
                        This is 0 is PowerVisor is on another screen
   138          4       IntuiMsg class
   142          2       IntuiMsg code
   144          4       IntuiMsg IAddress
   148          2       IntuiMsg MouseX
   150          2       IntuiMsg MouseY
   152          2       IntuiMsg Qualifier
   154          4       Pointer to Global

                          Offs  Size  Function
                          ---------------------------------------------
                          0     4     ln_Succ
                          4     4     ln_Pred
                          8     14    Physical window list
                          22    4     Pointer to active logical window
                          26    4     Signal set for all physical windows
                         <30    4     >

<  158          4       >
<  162          4       >
   166          18*6    For each logical window except 'Main'.
                        (Note that this area is saved with 'saveconfig',
                        so any changes you make here are permanent when
                        you make a config-file)

                        The first 8 bytes of each entry contain the NULL
                        terminated string used to open the logical window
                        (with the predefined command). The string can be
                        something like :
                          '0110d  ',0
                        which means :
                          go to master box
                          take child 0
                          take child 1 of this child
                          take child 1 of this child
                          take child 0 of this child
                          open logical window down this child
                        default strings are 'u      ',0

                        the word after this string is the share that this
                        window should take (in percentages x10)
                        default is 300.

                        After the 8 bytes for the string and the word
                        for the share value follows the 4 words for the
                        position for the physical window if mode 'intui'
                        is true. If these positions are used, the other
                        share variables are ignored ('mode intui').

                        (Extra,Debug,Refresh,PPrint,Rexx,Source)

   274          4       Startup flags (Updated and saved by 'saveconfig')
                          bit 0 : if true we open on workbench screen
                          bit 1 : if true we open on pv screen but with
                                  non-backdrop screen
   278          2*4     Four words describing the startup window (Updated
                        and saved by 'saveconfig').
```

			(x,y,w,h)
	286	2*2	Two words describing the startup screen (like the 'prefs screen' command). (w,h)
	290	24	24 pens (only 21 used at this moment) for fancy screens. (See the 'InstallingPowerVisor' file for more information about these pens.
	314	24	24 pens for no-fancy screens.
	338	4	Pointer to current pen table (one of the above tables, but you may make your own pen table and let this variable point to it)
	342	6	string with -BUSY- prompt
	348	6	string with -MORE- prompt
	354	6	string with -WAIT- prompt
	360	6	string with ------ prompt
	366	4	string with ???? prompt
	370	2	feedback prompt '> '
	372	4	the locked logical window (this is the logical window that is waiting for input)
	376	4	the pointer to the prompt string for the locked logical window
	380	1	lock state, the state of the stringgadget for the locked logical window (1 is no stringgadget, 0 is normal stringgadget)
	381	1	Busy mode (0 = normal, 1 = -BUSY-, 2 = a window is waiting for input)
*	382	1	GadgetExists. If 1 the stringgadget exists
<	383	1	>
	384	34	Fontname (default topaz.font), area is saved with the 'saveconfig' command
	418	4	Start of TextAttr. Pointer to 384:34 (fontname)
	422	4	Size of font (word), style (byte) and flags (byte) This area is saved with the 'saveconfig' command
	426	2	Height of all logical window borders
<	428	2	>
	430	2	Drag tolerancy at the left of the bar between two logical windows. This value indicates the amount of pixels at the left side of this line that PowerVisor will accept as the area used to drag this line
	432	2	Drag tolerancy at the top of the bar between two logical windows
	434	2	Drag tolerancy at the right of the bar between two logical windows
	436	2	Drag tolerancy at the bottom of the bar between two logical windows. This tolerancy value is normally larger (or just as large) as the height of the logical window border (426:2)
	438	2	Horizontal size tolerancy. This is the minimum width in pixels allowed for a logical window
	440	2	Vertical size tolerancy. This is the minimum height in pixels allowed for a logical window
	442	4	Pointer to requester structure allocated with rtAllocRequestA in reqtools.library
*	446	4	Pointer to reqtools.library (or 0 if no reqtools found)
*	450	4	Pointer to shared window port for IDCMP messages

```
*  454          4          Pointer to snap buffer (120 bytes). You may
                           use this buffer for other purposes. Note that
                           this buffer will be corrupted when the user snaps
                           something
```

1.14 The Wizard Corner : memory base (pvcall 41)

```
   Offset      Size       Function
-------------------------------------------------------------------------
   0           4          List with all automatic clear memory. You can
                          add things to this list with 'pvcall 49'.
                          This pointer is actually the pointer to the first
                          element in the list, and the first element in the
                          autoclear list is the LAST element that was added
                          (this is the oldest entry).
                          Each entry in this list contains a pointer to the
                          next entry (at offset 0) and a pointer to a PV
                          block (at offset 4).
   4           4          Pointer to the last element in the autoclear list
                          (previous list).
   8           2          The number of entries in the autoclear list.
   10          2          The maximum number of entries in the autoclear
                          list. You change this number (default 10) but you
                          must make sure that the list contains less entries
                          than the number you supply as a maximum.
   12          8          PV memory block containing the pointers to all
                          allocated memory (with the 'alloc' function and
                          the 'pvcall 55' command).
<  20          4          >
   24          8          PV memory block for the current tag list.
                          Format for one element in this PV memory block :

                            <Address>.L <Bytes>.L <Flags>.W <Type>.W
                            <Extra>.L

                               <Address> is start of memory block
                               <Bytes> is number of bytes for memory block
                               <Type> is one of

                                  1 = BYTEASCII BA
                                  2 = WORDASCII WA
                                  3 = LONGASCII LA
                                  4 = ASCII AS
                                  5 = CODE CO
                                  6 = STRUCT ST

                               <Flags> has the following bits

                                  bit 0 = write protect
                                  bit 1 = read protect
                                  bit 2 = ignore global protection
                                  bit 3 = print bus error information
                                  bit 4 = freeze (not supported yet)

                               <Extra> is the pointer to the structure
```

definition if <Type> = 6.

	32	4	Number of current default tag list (0..15)
	36	16*8	16 PV memory blocks containing all tag lists.
<	164	4	>
	168	4	Pointer to next memory to list (with 'memory', 'unasm' or 'view')
	172	4	Pointer to address to continue the search with 'next'.
	176	4	Remaining number of bytes to search.
	180	4	Pointer to string to search (PV block).
<	184	1	>
<	185	1	>
	186	8	PV memory block for the list containing all resident code fragments loaded with the 'resident' command
<	194	4	>
<	198	10	>
	208	4	Last number of bytes used with 'memory' or 'view' (default 320)
	212	4	Last number of lines used with 'unasm' (default 20)
	216	4	Pointer to first region for PowerVisor internal memory management. A region (or PV memory header) looks like this (it looks a bit like an Exec memory header) :

Offs	Size	Function
0	4	Next
4	4	Prev
8	4	Pointer to first free block in this region
12	4	Number of bytes free in this region
16	4	Pointer to start of region
20	4	Total size of region
24	4	Attributes for this region (like MEMF_CHIP)
<28	4	>