Your pal, WDEB386

Bill Gates was moderator at the annual Computer Trivia show, a Hollywood Squares-style fund raiser in San Jose which gets a lot of industry bigwigs together to compete on obscure computer questions with a lot of money being raised for charity. Gates posed the question: "There is a long-running contest on the Usenet to write the most confusing or bizarre, but working C Program. Name this contest."
Former Apple Computer executive Jean Louis-Gassee responded without hesitation "Microsoft Windows".

Dad was trying to play a Windows game with his six-year old son..
"Can I play it in the dark place?' asked the son.
Dad was unsure what his son meant until he exited out to DOS.
"That's the dark place!"

Dedicated to Brian Smith, who observed that if programming languages were sung, assembly would be a Gregorian chant.

WDEB386 is the primary tool for debugging Microsoft Windows 95. WDEB386's durability (hard to crash), flexibility (easy to extend) and low overhead (a terminal) are the major reasons why it remains top dog of the debuggers. This document provides a concise listing of the most important commands available through WDEB386 and sample output. There are three basic types of commands available: WDEB386 'dot' commands; VxD device driver commands; and WDEB386 direct commands. .dot and direct commands are usually available in both debug and retail builds. VxD device driver commands are in fact part of the specific VxD binaries, and change frequently and without warning. VxD commands are often only available in the component's debug version. This document was created on an EISA/PCI bus Pentium running full debug Windows 95, the RTM edition. Notes in italics are from developers, usually Briansm, Mikem or Raymondc. Note that the purpose of this document is to provide a handy resource to better enable you to debug applications under Windows 95, and not to provide a "help desk" for people with debugging problems. Please send comments or suggestions on this document to Percyt..

WDEB386 dot commands:

```
#.?
.? - prints help message
.B <baud rate> [<port addr>] - set COM baud rate/port addr (1 = COM1, 2 = COM2)
.REBOOT - reboots machine
..<cmd> - pass "cmd" directly to the VMM
.P - Dump scheduler data. Type '.P?' for more information.
.C - Dos Call trace information.
.M - dump memory manager structures.  Type '.M?' for more information.
.Y <expr> ---- Displays a CONFIGMG structure
.I* ---------- Dumps I/O Subsystem structures.  Type '.I?' for more info
.K - structure dumper.  Use .K? for more help.
.W - dump win32 data structures.  Type '.W?' for more information.
.H ----------- Display USER structures (.H? for help)
.A ----------- Display WINMM & MMSYSTEM structures (.A? for help)
.DM - Dumps the module list
.DQ - Dumps the task queue
.DG [handle | selector | arena(386)] - Dumps the global heap
.DH [0 (from top) | -1 (from bottom)] - Dumps the local heap
.DF - Dumps the free list
.DU - Dumps the LRU list
.CSP --------- Toggles catch stray pointer flag
.R [#] ------- Displays the registers of the current thread
.VM [#] ------ Displays complete VM status
.VC [#] ------ Displays the current VMs control block
.VH [#] ------ Displays a VMM linked list, given list handle
.VR [#] ------ Displays the registers of the current VM
.VS [#] ------ Displays the current VM's virtual mode stack
.VL ---------- Displays a list of all valid VM handles
.T  ---------- Toggles the trace switch
.S  [#] ------ Displays short logged exceptions starting at #, if specified
.SL [#] ------ Displays long logged exceptions just #, if specified
.LQ ---------- Display queue outs from most recent
.DS ---------- Dumps protected mode stack with labels
.VMM --------- Menu VMM state information
.<dev_name> -- Display device specific info
```

I got tired of not having any debugger commands for multimedia handles and stuff, so I took doozer and hacked it up into VWINMM.EXE. This is a self loading VXD like doozer, it adds the dot command 'A' with several sub options. It knows how to dump WinMM and MMSystem handle tables, WinMM driver table & Winmm Sound events table. if you have winmm & mmsystem symbols loaded, It knows how to grovel around to find the root of the handle tables. If you dont have symbols, you can find the handle tables the hard way, and then set them into the debugger with the .a@ command.

```
#.A ----------- Display WINMM & MMSYSTEM structures (.A? for help)
```

```
.aw*  xxxx Display WINMM core info (32bit)
.ah*  xxxx Display MMSYSTEM handle (16bit)
.ad*  xxxx Display Driver (codec) (32bit)
.ap*  xxxx Display PlaySound event (32bit)
.as*  xxxx Display PlaySound event (16bit)
.am*  xxxx Display MCI command tables (16bit)
.a@*  xxxx Show/Set base pointers (for working without symbols)
.ax*  xx   Set Debug Level
```

#.C - Dos Call trace information.

```
DOSTRC DOS trace level info.
   [1]  Display DOS trace profile
   [2]  Reset DOS trace profile
   [3]  Set trace level (currently 00)
   Any other key, return to previous menu.
Enter selection or [Esc] to exit DOSTRC debug query:
0 DOS calls have been profiled

DOSTRC DOS trace level info.
   [1]  Display DOS trace profile
   [2]  Reset DOS trace profile
   [3]  Set trace level (currently 00)
   Any other key, return to previous menu.
Enter selection or [Esc] to exit DOSTRC debug query:

Level 0 = DOS tracing turned off
Level 1 = Only LOG # times each call is made
Level 2 = Log # times each call is made + total time for calls
Level 3 = Log time for calls + dump ASCIIZ strings to debug terminal
Level 4 = Dump complete profile data to monochrome display for each call
Level 5 = Same as 4 except stops before reflecting call to V86 mode
Level 6 = Same as 4 except stops after call returns from V86 mode
Level 7 = Same as 5 and 6 combined
Use system.ini variable DosTraceLevel=X to do this automatically (default=0)
Enter new level or [Esc] to quit:
```

#.DF - Dumps the free list

```
Address     Arena    Size Type Owner    Handle  LRU Links    Seg|Rsrc
    B840:      80      0b SENTINEL              FFFFFFFF,3B40
8016BEC0:    3B40     A0b FREE                    80,4300
80233EE0:    4300    C80b FREE                  3B40,45E0
80235CE0:    45E0    660b FREE                  4300,43E0
80233600:    43E0    440b FREE                  45E0,4520
80233300:    4520     C0b FREE                  43E0,4380
80233080:    4380     40b FREE                  4520,3E40
80160000:      E0      0b SENTINEL              2840,FFFFFFFF
```

#.DM - Dumps the module list

```
hExe Type ModName  File Name
0117 DYNA KERNEL   C:\WINDOWS\SYSTEM\KRNL386.EXE
01DF DYNA SYSTEM   C:\WINDOWS\SYSTEM\system.drv
010F DYNA KEYBOARD C:\WINDOWS\SYSTEM\keyboard.drv
01E7 DYNA MOUSE    C:\WINDOWS\SYSTEM\mouse.drv
01FF DYNA DISPLAY  C:\WINDOWS\SYSTEM\atim32.drv
0387 DYNA DIBENG   C:\WINDOWS\SYSTEM\DIBENG.DLL
0257 DYNA SOUND    C:\WINDOWS\SYSTEM\mmsound.drv
0307 DYNA COMM     C:\WINDOWS\SYSTEM\comm.drv
044F DYNA GDI      C:\WINDOWS\SYSTEM\gdi.exe
180F DYNA FONTS    C:\WINDOWS\fonts\vgasys.fon
1817 DYNA FIXFONTS C:\WINDOWS\fonts\vgafix.fon
1807 DYNA OEMFONTS C:\WINDOWS\fonts\vgaoem.fon
17DF DYNA USER     C:\WINDOWS\SYSTEM\user.exe
16CF DYNA DDEML    C:\WINDOWS\SYSTEM\DDEML.DLL
16C7 DYNA SERIFE   C:\WINDOWS\fonts\serife.fon
14DF DYNA SSERIFE  C:\WINDOWS\fonts\sserife.fon
16D7 DYNA COURE    C:\WINDOWS\fonts\coure.fon
16A7 DYNA SYMBOLE  C:\WINDOWS\fonts\symbole.fon
169F DYNA SMALLE   C:\WINDOWS\fonts\smalle.fon
1697 DYNA MODERN   C:\WINDOWS\fonts\MODERN.FON
1627 NAME MSGSRV32 C:\WINDOWS\SYSTEM\MSGSRV32.EXE
1F7F DYNA MMSYSTEM C:\WINDOWS\SYSTEM\mmsystem.dll
1F87 DYNA LZEXPAND C:\WINDOWS\SYSBCKUP\LZEXPAND.DLL
1F77 DYNA VER      C:\WINDOWS\SYSBCKUP\VER.DLL
1ED7 DYNA SHELL    C:\WINDOWS\SYSTEM\SHELL.DLL
1DD7 DYNA COMMCTRL C:\WINDOWS\SYSTEM\COMMCTRL.DLL
```

```
1CCF DYNA NETWARE  C:\WINDOWS\SYSTEM\netware.drv
1A97 NAME SCOUT    C:\WINDOWS\SCOUT.EXE
1A4F DYNA SCOUTAPI C:\WINDOWS\SCOUTAPI.DLL
2927 DYNA PIFMGR   C:\WINDOWS\SYSTEM\PIFMGR.DLL
28E7 DYNA DOSAPP   C:\WINDOWS\fonts\dosapp.fon
```

#.DQ - Dumps the task queue

```
  TDB   SS:SP   nevt prty  hQ  Module
 1C66 0000:0000 0000 0000 1C1F EXPLORER
 1D5E 0000:0000 0000 0000 1CF7 MPREXE
 163E 1FFF:33EA 0000 0000 2097 MSGSRV32
 1A7E 1A57:4734 0000 0000 1A0F SCOUT
*0097 0137:1258 0000 0000 2097 KERNEL
```

#.DS ---------- Dumps protected mode stack with labels

```
0030:C1A11D84 = C0123A3C, C0123928:DEBUG(01) + 114
0030:C1A11D88 = C0005456, TO_Dispatch_Return
0030:C1A11D8C = C163AB70, 0028:C163AB70
0030:C1A11D90 = C0004601, End_Process_Event
0030:C1A11D94 = 00000000, 0028:00000000
0030:C1A11D98 = C163A5C4, 0028:C163A5C4
0030:C1A11D9C = 00000000, 0028:00000000
0030:C1A11DA0 = C0001400, Return_To_VM
0030:C1A11DA4 = 00000030, 0028:00000030
0030:C1A11DA8 = 00000030, 0028:00000030
0030:C1A11DAC = C1A11DD0, 0028:C1A11DD0
0030:C1A11DB0 = C163A4B0, 0028:C163A4B0
0030:C1A11DB4 = C1A11F70, 0028:C1A11F70
0030:C1A11DB8 = C1A11DCC, 0028:C1A11DCC
0030:C1A11DBC = C1D20154, 0028:C1D20154
0030:C1A11DC0 = 00000000, 0028:00000000
```

#.DU - Dumps the LRU list

```
Address     Arena    Size Type Owner    Handle  LRU Links    Seg|Rsrc
801E8500:    3900    B60b PRIV KERNEL  (014E,1)  4A0,4580
81537B20:    4580   6F00b CODE PIFMGR  (283E,1) 3900,3A60    2
81559BE0:    3A60   3040b CODE PIFMGR  (2876,1) 4580,4400    1
80232D80:    4400    300b RSRC PIFMGR  (284E,1) 3A60,4480    Icon 2
```

.Doozer

Finally fed up with the lack of proper debugging tools for USER, I wrote my own. Doozer adds the following:

```
        .hwnd   -- Dump a window
        .hcls   -- Dump a window class
        .hsms   -- Dump a sms
        .h?     -- Duh...
```

All of the above commands accept handles, based pointers, or variables, and they support an `interactive mode' which lets you traverse the window / class / sms lists quickly. For example...

```
        13##.hwnd hwndtty,            <- View the window whose handle is stored in the variable
                                         hwndtty and enter interactive mode
        Window 20bf0 (01cc) "MS-DOS Prompt - vi DOOZER.ASM" (tty)
          N=20e10 C=20c58 P=20118 O=    0
          W=(-32000,-32000)-(-31894,-31976)  C=(0,0)-(0,0)
          hq=0e47 pcls=46d4 hinst=0e2e wp=0d37:00000bb4
          162C 8113 3844 8113
                [Top line gives the window pointer, handle, title, and class.
                 N = Next, C = Child, P = Parent, O = Owner
                 W = window rectangle, C = client rectangle
                 wp = window procedure; the four words at the end
                 are a dump of the window's extra bytes]
        C                             <- Go to this window's Child window
        Window 20c58 (01d0) "" (ttyGrab)
          N=20f3c C=    0 P=20bf0 O=    0
          W=(0,0)-(0,0)  C=(0,0)-(0,0)
          hq=0e47 pcls=4704 hinst=0e2e wp=0d37:00000ba8
        N                             <- Go to this window's Next window
        Window 20f3c (01d4) "" (ToolbarWindow)
          N=    0 C=21004 P=20bf0 O=    0
          W=(0,0)-(0,29)  C=(0,2)-(0,29)
          hq=0e47 pcls=3434 hinst=130e wp=12c7:0000005a
          0824
        P                             <- Go to this window's Parent window
        Window 20bf0 (01cc) "3" (tty)
          N=20e10 C=20c58 P=20118 O=    0
          W=(-32000,-32000)-(-31894,-31976)  C=(0,0)-(0,0)
          hq=0e47 pcls=46d4 hinst=0e2e wp=0d37:00000bb4
          162C 8113 3844 8113
        <ESC>                         <- Exit interactive mode
```

doozer.exe is another one of those triple-mode files: It's a DOS app, a static VxD, and a dynamic VxD all rolled into one. Install it statically by putting `device=doozer.exe' in your system.ini; or you can install it dynamically by typing `doozer' at a DOS prompt. If you loaded it dynamically, you can unload it by typing `doozer' a second time. Doozer.exe is installed automatically in the Windows\system directory by PSGRET when symbols are installed, and comes free with debug Windows 95.
`.hq 0' to dump the list of pending PostMessages. (For when you hit that `int 3' deep in the bowels of USER.)

#.H? ----------- Display USER structures (.H? for help)

```
.hwnd xxxx --- Display a window
.hcls xxxx --- Display a class
.hsms xxxx --- Display a SendMessageStructure
.hvwi xxxx --- Display a vwi
.hq xxxx ----- Display a queue
.hmenu xxxx -- Display a menu
.hatm xxxx --- Display a global atom
.hicon xxxx -- Display an icon or a cursor
.hprop xxxx -- Display a property list
.hbsm -------- Diagnoses a hung BroadcastSystemMessage
.hf ---------- Toggle `full' (verbose) mode

Arguments can be handles, based pointers, or variables whose values
are in turn handles or based pointers.  Appending a comma enters
interactive mode; see command-specific help (`.hwhatever ?').

Examples:
.hwnd 150,     Dump window whose handle is 0150 and go interactive
.hwnd 20ce4    Dump window whose address is 20ce4
.hwnd hwndKbd  Dump window whose handle/address is in the variable hwndKbd
               Doozer will try to guess whether it's a handle or address
```

#.HWND 150

```
Window 21D90 (0150) "" (4)
  N=21D28 C=    0 P=205E8 O=    0
  W=(0,0)-(112,27)  C=(3,22)-(109,24) pmenu=0000:00000000
  hq=1C3F pcls=3C84 hinst=1BB7 wp=141F:000001C4 -> %7EB4493C
  0020 A000 6ABA BFF7 0030 4842 73CF 40C3
```

#.I? ---------- Dumps I/O Subsystem structures. Type '.I?' for more info

```
*** I/O Subsystem debug help ***
.IIOP <exp> - dumps IOP/IOR/IOR callback structures for specified address
.IDCB <exp> - dumps DCB and the calldown table lists for specified address
.IMED       - lists IOS Memory Pool entries
.ISP <exp>  - dumps the specified number of SCSI trace records
.ISP <exp>  - dumps the specified number of SCSI trace records
.IDV <exp>  - dumps the specified number of SCSI DISK VSD trace records
.IVT <exp>  - dumps the specified number of Voltrack trace records
```

#.K? - structure dumper. Use .K? for more help.

```
Scanning for GDI symbols etc, please wait...done.
Generic structure dumping commands
  .kl - lists available structures
  .ks strucspec - sets the default structure to dump with
  .kd <[strucspec]> addr - dumps structure at addr.

  strucspec: strucname<.startfield<-endfield>>
             specifies structure name to use and optional starting/ending
             fields within the structure to use.  Each of these entries can
             end in a '*' which means search for a matching prefix.  If a
             prefix is not unique a menu will be displayed.  See examples
             section.
  Note: .kd does NOT assume addr is relative to ds - you must type in seg.

GDI-specific dumping commands
  .kg - enables/disables GDI-specific stuff
  .ka atom - prints the string for the specified atom from GDI's atom heap
  .kc objecttype - dumps all existing objects of specified type (summary)
  .kcv objecttype - same as .kc but dumps entire structure for each obj too
  .ksdc <hdc> - sets current driver for .kddi.  If no parm use screen DC
  .kddi - dumps DDI parm list.  cs:ip must be at entry point of driver
  .kdl - dumps DC swapping log
  .kf <hdc> - dumps driver entry points for hdc (or screen if none specified)
  .kt - menu of settings available for DEBUG GDI

Structure 'Intellisense'
  VDBSTRUC looks for signatures that identify the object type and will use
  an appropriate structure type based on the object.  This works for:
    - logical GDI objects
    - DIB Engine PDevice
    - DDI call parameters

Examples
  .kd 567:294a - dump structure at 567:294a
  .kd [gdiinfo] 567:2980 - dump 567:2980 using the gdiinfo structure
  .kd [gdii*] 567:2980 - same as above, * means match prefix to strucname
  .ks dis*.hMeta*-dchpal - sets default structure to DisplayContext,
                           displaying fields from hMetaFile to dchPal.
  .ka c342 - displays string for atom c342
  .kc brush - displays all brushes in system
  .kcbr - displays all brushes.  Note: unique prefix works, no space needed
  .kc parameters: PEn, BRush, Font, PAlette, BItmap, Region, Dc, Metafile, Emf
```

#.LQ ---------- Display queue outs from most recent

```
00000000: [0000000E] VCD_Detach COM02 from C1D20154
00000000: [00000009] VCD_Detach COM01 from C1D20154
00000000: [00000009] VCD_Attach COM02 to C1D20154
00000000: [00000009] VCD_Attach COM01 to C1D20154
00000000: [00000001] VCD_Detach COM02 from C1D20154
00000000: [00000001] VCD_Detach COM01 from C1D20154
00000000: [00000001] VCD_Attach COM02 to C1D20154
00000000: [00000001] VCD_Attach COM01 to C1D20154
```

.MW

The `.mw' command now accepts an optional vp index which indicates where the dump should begin. (The default still is to begin dumping at the beginning of the array.) If you're experiencing a memory leak, you can dump the last few committed pages to see who allocated them.

#.M - dump memory manager structures. Type '.M?' for more information.

```
.M[<struct>][<link>] [<expression> [L<count>]]
 where <struct> is a single character to specify the structure to dump
 (default is to dump all the structures assicated with a given linear address):
  'A' - AR - Arena Record              'P' - PF - Page (Physical) Frame
  'C' - CD - Context Descriptor        'R' - AH - Arena Header
  'D' - PDE - Page Directory Entry     'T' - PTE - Page Table Entry
  'G' - PD - Pager Descriptor          'V' - VP - Virtual Page
  'H' - Heap Block or Heap Handle      'X' - All structures for an address
  'L' - Phys-Linear mapping areas      'Y' - Valid physical memory ranges

  If <struct> is not specified, a summary about the specified page is printed.

 <link> is single character to specify how to walk the structures if
 dumping more than one (default is to walk adjacent entries):
  'B' - backward link
  'F' - forward link

 <expression> is a debugger expression that identifies what structures to
 dump (a linear address of the structure, or handle, or array index, etc...).
 For .m and .mx, the default is the contents of the cr2 register (the last
 page fault) and for the other commands it is the first structure in the list.

For example, '.M' will give you information about the page most recently
faulted upon.  '.MPF EAX L10' will dump the ten PFs starting at the linear
address in EAX and follow their forward links

In addition to the above, there are other memory manager commands that do not
dump structures:
  'E'  - Toggle stopping in the debugger for memory manager errors
  'F'  - With no arguments, dump page fault log.  With arguments, set to log
         faults in a specific range (syntax: .MF <address> L<lengh in bytes>).
  'FB' - Toggle stopping for eached logged page fault (see above)
  'FF' - Toggle logging page faults to debug terminal
  'HS' - Dump statistics about a given heap (syntax: .MHS [<heap handle>])
  'HW' - Toggle paranoid VMM32 heap checking
  'HX' - Check for corruption on given heap (syntax: .MHX [<heap handle>])
         defaults to VMM32 fixed heap.
  'I'  - Dumps info about instance data
  'M'  - Force memory present (syntax: .MM [<address> [l<page count>]] )
```

```
         <address> defaults to last page faulted upon (contents of cr2 register)
  'N'  - Force all unlocked memory not present
  'S'  - Dump memory manager statistics
  'U'  - Toggle computing checksums for page-outs and page-ins
  'W'  - Dump allocators of all committed pages
```

A word of caution about the .MM command -- if you invoke it at the wrong time, you can crash your system. The only time it can be guaranteed safe is if you are currently executing ring 3 code. At other times we may be in the middle of another file system or memory manager operation. This warning also applies to .MHX and .MHS

#.M - dump memory manager structures

```
PageBase=BFF28000 committed r/o user   EXE/DLL present  PhysAddr=016F0289
 ObjBase=BFF20000 cpg=0002F PID=0001 Owner=BFF20000:GDI32.DLL
```

#.MS - Dump memory manager statistics

```
Physical pages:   01F87  Free phys pages: 00ABA  Idle phys pages:  00000
Committed pages:  01670  Swappable pages: 0053B  Other type pages: 01135
Of those, Locked: 00E93  Maximum locked:  01F57  Unsafe pages:     00000
Disk cache pages: 00900  Minimum cache:   0011E  Maximum cache:    01952

Swapping ON:  Paging through Dragon
Swap file size (in pages): 00000 In use: 00000 Needed: 00000 Broken: 00000
Highest in use: 00000 Running total allocated: 00000000 Stolen: 00000000
Minimum size:   00000 Max in system.ini: 80000 Available max at boot: 18ADC
Swap file name: C:\WINDOWS\WIN386.SWP

Page faults: 000005A3 Page-In: 00000D5D Page-Out: 00000000 Discard: 00000000
Instance:    00000000 Hooked:  00000010 Nul page: 00000000 Invalid: 00000013
Reclaims:    00000000                            Stupid page-outs: 00000000
Page tables outs: 00000000 Page table discards: 00000000
```

The `.mwf' command displays all locked pages and their owners. (I wish I had this command last week.) Playing around with it, I noticed that some people are allocating permanently-locked memory by calling PageAllocate(...,PageLocked) instead of PageAllocate(..., PageFixed). This wastes swap space, so don't do that.

#.P? - Dump scheduler data. Type '.P?' for more information.

```
The following commands are available through .p:
        For more extensive help on a command type .p<cmd>?
        .p  - lists threads in system
        .p <*/thread id> - lists status of one thread
        .pf - lists threads and their flags
        .ps - <Thread handle/id> - Dumps ring 0 stack with labels
        .psx - <Thread handle/id> - Dumps 20 lines of ring 0 stack and returns
        .pdev <Address> - Finds nearest VXD name
        .plog <flags> - set scheduler query logging flags
        .pmtx <Mutex Address> - display mutex state
        .psem <Semaphore Address> - display semaphore state
        .pthcb <Thread handle/id> - display thread control block
        .pprd - Disables the logging of priority changes
        .ppre - Enables the logging of priority changes
        .pprf <filter> - Logs only boosts changing these bits
        .pprl <Thread handle/id> - Lists priority changes recorded
        .pmax - Show thread and VM maximum DOS386 stack usage
```

#.P - Dump scheduler data

```
 Th VM THCB     ThFlgs   VMFlg  VMPri    ThPri    Evnt State
 13 01 C1F9A8AC 00511401 001842 00000000 00000009  Yes Blockd on SEM: C1F9A81C
 11 01 C1F99860 00510401 001842 00000000 00000009  Yes Blockd on SEM: C1F996E8
 10 01 C1F98910 00510401 001842 00000000 00000009  Yes Blockd on SEM: C1F984D4
 0B 01 C1F96478 00511401 001842 00000000 00000008  Yes Blockd on SEM: C1F963E8
 0A 01 C1F8FA80 00510401 001842 00000000 0000101E  Yes Blockd on ???: 00000000
 09 01 C1F8F788 00511401 001842 00000000 00000009  Yes Blockd on SEM: C1F8F6F8
 08 01 C1F55830 00510401 001842 00000000 0000000F  Yes Blockd on SEM: C1E93340
 06 01 C1F8D5D8 10110401 001842 00000000 0000000C  Yes Blockd on ???: 00000000
 05 01 C1F8D338 10110401 001842 00000000 0000000C  Yes Blockd on ???: 00000000
 04 01 C1E87E34 10110401 001842 00000000 0000000C  Yes Blockd on ???: 00000000
 03 01 C1E910E8 10110401 001842 00000000 0000000C  Yes Blockd on ???: 00000000
 02 01 C17391F4 00510401 001842 00000000 0000000D  Yes Blockd on SEM: C1E8C520
*01 01 C1D9F08C 00010000 001842 00000000 00100004      Ready
```

I added a new debug command, .pdev. When a VXD crashes this command can be used to dump the name of the VXD that caused the problem. It simply does automatically what we usually do manually. It looks through the list of VXDs loaded, looking for the nearest VXD control table.

#.P?DEV - display VXD at specified address

```
Command: pdev <addr>
Syntax:  <addr> - Address to try and match to a VXD.
Description:
        Attempts to match an address to a VXD name. It does
        this by searching the VXD list for the closest VXD control
        proc that has an address smaller than the address passed in.
```

#.PDEV EIP - display VXD at specified address

```
DEBUG(Locked Code) + 149
```

#.VC [#] ------ Displays the current VMs control block

```
Dump of VM control block for VM handle C1D20154
0000003C  hVM                  C1D20154
00000000  Flags                00005842
00000004  High Linear Addr     C1C00000
00000008  Client_Pointer       C1A11F70
0000000C  VMID                 00000001
FFFFFFEC  ppteVM               FFB07000
FFFFFF8C  hheapVMCB            C1D20000
00000030  VM_List_Link         00000000
FFFFFFE8  Suspend_Count        00000000
FFFFFFF0  pdeVM                0053A267
00000038  Inst_Buf_ptr         C1D2164C
00000028  MMGR_Flags           00000005
0000001E  LDTSel               00B8
FFFFFFC8  LDTcpg               0003
0000002C  LDTAddr              80098000
FFFFFFF8  Int_Enable_Count     FFFFFFFF
00000018  VM_Exec_Time         007B087E
FFFFFFE0  ThreadListHead       C1E7F854
00000020  VMM_Flags            00000008
FFFFFFE4  TS_Sched_Count       00000000
FFFFFFDC  pthcbThread          C1D9F08C
FFFFFFE0  ThreadListHead       C1E7F854
```

#.VH [#] ------ Displays a VMM linked list, given list handle

```
Usage:  .VH[B|W|D] hlist
  B|W|D = Dump as bytes, words or dwords (default=dwords)
  hlist = VMM list handle
```

#.VL ---------- Displays a list of all valid VM handles

```
*C1D20154, id 0001, status 00005842
        Back    PM_App  Awake   PgV86   IdleTOut
 C4920154, id 0002, status 00008802
        Back    Awake   Idle
```

#.VM [#] ------ Displays complete VM status

```
Windows/386 kernel reentered 0001 times
Critical section claim count = 00010000
Thread owning critical section = 00000000

VM handle      = C1D20154
Client pointer = C1A11F70
VM Status      = 00005842
        Back    PM_App  Awake   PgV86   IdleTOut

Alternate Register Set
AX=00001607  BX=C1F90018  CX=00020000  DX=0000001B  SI=00000097  DI=00001A7E
IP=000086B2  SP=00001236  BP=00000000  CR2=00000080  CR3=00508  IOPL=0  F=-- --
CS=011F  SS=0137  DS=0137  ES=0137  FS=0147  GS=0000 -- NV UP EI PL ZR NA PE NC
011F:000086B2  POP     DS

Registers for C1D20154
AX=00001607  BX=C1F90018  CX=00020000  DX=0000001B  SI=00000097  DI=00001A7E
IP=000003CC  SP=0000023A  BP=00000000  CR2=00000080  CR3=00508  IOPL=0  F=-- VM
CS=061E  SS=0B59  DS=95A4  ES=0B59  FS=0000  GS=0000 -- NV UP DI PL ZR NA PE NC
&061E:000003CC  CMP     AX,4B20

0B59:023A  0389 8966 0E73 33E8 FFF2 E9FF FF67 FFFF
0B59:024A  C1FB 02EE 8346 40FE 820F 0118 0000 FA5E
0B59:025A  1D8B B7CB C003 DB0B 2374 3966 0E73 0872
0B59:026A  F766 1043 0014 0574 5B8B EB04 66E9 4B83
0B59:027A  0410 8966 0C4B 038B B1E9 0000 F600 2444
```

```
0B59:028A  0110 850F 0181 0000 1D8B B5A8 C003 7B83
```

.VMM [C] Display blocked thread information

Starting in build whatever, `.vmm c' in retail or debug will list all threads which are blocked, who is responsible for blocking, and what they are blocked on.  If a thread is blocked on multiple objects, it will show all objects.  (This is different from `.p' which often says that a thread is Ready when in fact it is blocked.) The code which extracts this information is incredibly scary, but I figure it's worth the fright if it saves me an hour of debugging in other people's offices per day.

Real-world sample output of `.vmm c':

```
        thid obj Handle   Stack    Caller   Symbol (if known)
        0008 CS  C001F3C4 81B69E20 C02201EF DOSMGR_Int_21_Mapper + 53
        0006 sem C1651F5C 81933E3C C0066F01 _BlockThreadSetBit + 66
        0005 sem C16503E4 81831E48 C0066F01 _BlockThreadSetBit + 66
        0004 sem C164ED64 818CFE3C C0066F01 _BlockThreadSetBit + 66
        0003 sem C164ED64 8179DE54 C006702A _WaitForEvent + 24
        0002 sem C1635388 8161BE28 C0066F01 _BlockThreadSetBit + 66
        0011 ID  C37D2EE8 81B46CF0 C0069AEE _VBlock + 12
        000E CS  C001F3C4 81B3AE68 C02A0D36 DOSMGR_Int_2A + 62
        0010 sem C165E250 81B43E3C C0066F01 _BlockThreadSetBit + 66
        000D sem C165A3BC 81B37E3C C0066F01 _BlockThreadSetBit + 66
        0009 sem C1658724 81B35E28 C0066F01 _BlockThreadSetBit + 66
```

From this, you can see that threads 0008 and 000E are both blocked in DOSMGR on the critical section (CS). If you type `.vmm b' you would see that the critical section owner is thread 0011, which we can see from the above output is blocked in IFSMGR on ID C37D2EE8.

So the problem is that an I/O request issued by IFSMGR is not completing. To determine the reason for this I/O request, you type `ds 81B46CF0' to dump the stack of the IFSMGR BlockOnId. From this, you see `_ReadWrite + 3f' and `_CatchReq + 159', which mean that we are handling an I/O request issued from an application.  Typing `.vr 11' shows ax=3DA0, which indicates an `open' call.

So what happened is that the application tried to open a file, which got routed through IFSMGR, which issued an I/O request and then blocked waiting for the completion, but the completion never came back. Meanwhile, other people wanting to do I/O have queued up and are waiting for IFSMGR to release the critical section. The next step, of course, is to dump the IOS private heap to locate the errant I/O request. But that takes us beyond the scope of the `.vmm c' command; the point was to illustrate how `.vmm c' lets you zero in on the deadlock quickly without ever having to call RaymondC into your office.

One item not illustrated by the above example is the case where a thread is blocked on multiple objects. In such a case, you will see multiple lines with the same `thid' entry. They are listed in LIFO order. (The most recent block for a thread appears at the top of the list.)

For completeness' sake, here's what each column means, if you haven't figured it out already.

```
        thid = thread ID
        obj  = object type being blocked on
        Handle = object handle
        Stack = What to `ds' to investigate further
        Caller = the code that issued the Begin_Critical_Section,
                Wait_Semaphore, BlockOnID, etc.
        Symbol = symbol for caller, if symbols are loaded

        Object types and corresponding handles:
                mtx = mutex (handle = mutex handle)
                sem = semaphore (handle = semaphore handle)
                ID  = BlockOnID (handle = ID being blocked on)
                CS  = Critical section (handle = critical section handle)
                V86 = V86 mutex (handle = V86 mutex handle)
```

.VMM L=DEVNAME

You can say

.vmm l=devname

to list a particular device by name. For example, `.vmm L=int13'. This lets you answer questions like `Is Foo.vxd loaded? If so, where?' without having to page through dozens of VxDs looking for it and hoping you didn't miss it in the list.

#. VMM --------- Menu VMM state information

```
         V M M   D E B U G   I N F O R M A T I O N A L   S E R V I C E S
[A]  System time
[B]  Display Critical Section info
[C]  Display blocked thread information
[D]  Reset dyna-link profile counts
[E]  I/O port trap information
```

```
[F]  Reset I/O profile counts
[G]  Turn procedure call trace logging on
[H]  V86 interrupt hook information
[I]  PM interrupt hook information
[J]  Reset PM and V86 interrupt profile counts
[K]  Display event lists
[L]  Display device list
[M]  Display V86 break points
[N]  Display PM break points
[O]  Display interrupt profile
[P]  Reset interrupt profile counts
[Q]  Display GP fault profile
[R]  Reset GP fault profile counts
[S]  Toggle verbose device call trace
[T]  Dyna-link service profile information
[U]  Fault Hook information
[V]  Display time out queues
[W]  PM CLI/STI trace info
[X]  DPMI info
```

Enter selection or [ESC] to exit: C

```
Display blocked thread information:
 thid obj Handle   Stack    Caller   Symbol (if known)
 0013 sem C1F9A81C C15CCE34 C00788C4 _BlockThreadSetBit + A2
 0011 sem C1F996E8 C15B6E18 C00788C4 _BlockThreadSetBit + A2
 0010 sem C1F984D4 C15A3E24 C00788C4 _BlockThreadSetBit + A2
 000B sem C1F963E8 C1599E18 C00788C4 _BlockThreadSetBit + A2
 000A ID  C1F8FA80 C1596E58 C022C226 _W32_BlockOnID@16 + 9
 0009 sem C1F8F6F8 C1593E34 C00788C4 _BlockThreadSetBit + A2
 0008 sem C1E93340 C1590E50 C0078A7A _WaitForEvent + 24
 0006 ID  C1E7F30C C154AD9C C1F42701 C1F398E0:VSERVER(01) + 8E21
 0005 ID  C1E92E20 C1547D9C C1F42701 C1F398E0:VSERVER(01) + 8E21
 0004 ID  C1F50770 C1544D9C C1F42701 C1F398E0:VSERVER(01) + 8E21
 0003 ID  C1F4DA78 C1540D9C C1F42701 C1F398E0:VSERVER(01) + 8E21
 0002 sem C1E8C520 C153DE44 C00788C4 _BlockThreadSetBit + A2
*0001 TO  00000000 C1A11D88 C0123A17 C0123928:DEBUG(01) + EF
*0001 NE  C163AA58 C1A11DA4 C00BBBA3 VMPoll_System_Idle + 2E
```

#.VR [#] ------ Displays the registers of the current VM

```
Alternate Register Set
AX=00001607  BX=C1F90018  CX=00020000  DX=0000001B  SI=00000097  DI=00001A7E
IP=000086B2  SP=00001236  BP=00000000  CR2=00000080  CR3=00508  IOPL=0  F=-- --
CS=011F  SS=0137  DS=0137  ES=0137  FS=0147  GS=0000 -- NV UP EI PL ZR NA PE NC
011F:000086B2  POP     DS

Registers for C1D20154
AX=00001607  BX=C1F90018  CX=00020000  DX=0000001B  SI=00000097  DI=00001A7E
IP=000003CC  SP=0000023A  BP=00000000  CR2=00000080  CR3=00508  IOPL=0  F=-- VM
CS=061E  SS=0B59  DS=95A4  ES=0B59  FS=0000  GS=0000 -- NV UP DI PL ZR NA PE NC
&061E:000003CC  CMP     AX,4B20
```

#.VS [#] ------ Displays the current VM's virtual mode stack

```
Thread Stack C1D9F08C
0B59:023A  0389 8966 0E73 33E8 FFF2 E9FF FF67 FFFF
0B59:024A  C1FB 02EE 8346 40FE 820F 0118 0000 FA5E
0B59:025A  1D8B B7CB C003 DB0B 2374 3966 0E73 0872
0B59:026A  F766 1043 0014 0574 5B8B EB04 66E9 4B83
0B59:027A  0410 8966 0C4B 038B B1E9 0000 F600 2444
0B59:028A  0110 850F 0181 0000 1D8B B5A8 C003 7B83
```

#.W? - dump win32 data structures. Type '.W?' for more information.

```
.W [<expression>]
where <expression> is the address of a win32 object of one of the following
types:
        thread          process         semaphore
        event           mutex           critical section
if <expression> is omitted, information on all threads is displayed.

.WHE  Toggle stopping in debugger for Win32 memory manager error returns
.WHW  Toggle Win32 paranoid heap corruption checking
.WL   Dump LDT-related info (eg. selman lists, sel free lists)
.WM   Dump module table
.WMP [ppdb]
Dump module table for process (no ppdb dumps current process)
.WP [ppdb]
Dump process list
```

```
.WC context
Dump context record
.WE exception
Dump exception record
.WD dispatcherContext
Dump dispatcher context
.WS  displays the status of all ring 3 system critical sections.
.WT [ppdb] or nothing for current process
Dump process handle table
If any data item being dumped resides in not-present memory, then
the address of that data item is displayed in brackets ("[]").
```

#.W - dump win32 data structures

```
 id  ptdb     ppdb     r0thcb   ptdbx    tdb  name         state
*1   81681360 81681218 C1D9F08C C1F60C80 0097 KERNEL32.DLL ready
 2   81582FD0 81681218 C17391F4 C1E8C494 0097 Krnl Service blocked
 3   81583608 81681218 C1E910E8 C1739A0C 0097 KERNEL32.DLL ready
 4   815841B0 81681218 C1E87E34 C1E87D18 0097 KERNEL32.DLL ready
 5   81584558 81681218 C1F8D338 C1E88030 0097 KERNEL32.DLL ready
 6   81584900 81681218 C1F8D5D8 C1F8D4BC 0097 KERNEL32.DLL ready
 8   815854E0 81681218 C1F55830 C1F55714 0097 KERNEL32.DLL ready
 9   81585CE8 81585900 C1F8F788 C1F8F66C 163E MSGSRV32(16) blocked
 A   81586090 81681218 C1F8FA80 C1F8F964 0097 Fault Thread blockonid: C1F8FA80
 B   81587154 81586CD4 C1F96478 C1F9635C 1D5E MPREXE.EXE   blocked on: 81588E94 81588B34
 10  8158B0DC 8158AC70 C1F98910 C1F980CC 1C66 EXPLORER.EXE blocked on: 8158A57C
 11  8158E0AC 8158AC70 C1F99860 C1F9976C 1C66 EXPLORER.EXE blocked on: 8158A7F8
 8158A848 8158E75C 8158FAA8 8158FB24 8158A784
 13  81590040 8158FBDC C1F9A8AC C1F9A790 1A7E SCOUT   (16) blocked
```

#.W 81587154

```
id  ptdb     ppdb     r0thcb   ptdbx    tdb  name         state
 B   81587154 81586CD4 C1F96478 C1F9635C 1D5E MPREXE.EXE   blocked on: 81588E94
 81588B34
              Id: FFFFECD7
            Type:        6 Thread
       Ref count:        1
        pVExcept:   72FCD8
    Top of stack:   730000
   Base of stack:   72E000
         K16 TDB:     1D5E
     Stack sel16:     1D47
     Selman list:        1
    User pointer:        0
            pTIB: 81587164
       TIB flags:     0001 TIBF_WIN32
Win16Mutex count:     FFFF
   Debug context:        0
  Ptr to cur pri: C1F96450 pri: 8
   Message queue:     1CF7
      pTLS array: 815871EC
           Flags: 00000008 fGrowableStack
          Status:      103
         TIB sel:     1D6F
    Emulator sel:     0000
    Handle count:       00
 APISuspendCount:        0
  Wait node list: C17C7504
      R0 hThread: C1F96478
      Stack base:   620000
 Terminate stack:   730000
   Emulator data:        0
     Debugger CB:        0
     Debugger TH: FFFFFFFF
         Context:        0
   Except16 list:        0
   Thunk connect:        0
  Neg stack base: FF8D9000
      Current SS:     1D47
        SS Table: 81587490
      Thunk SS16:     1D67
       TLS Array: 815871EC
  Delta priority:        0
      Error Code:        0
   pCreateData16:        0
          wSSBig:        0
   lp16SwitchRec:        0:0
           Stack: Normal
      Rip string:        0
       WOW chain:        0
           ptdbx: C1F9635C
   ContextHandle: C1F962B0
   TimeOutHandle:        0
       WakeParam:        0
     BlockHandle: C1F963E8
      BlockState: FFFFFFFF
    SuspendCount:        0
   SuspendHandle:        0
   MustCpltCount:        1
     WaitExFlags: 00000040 WIN32_NPX
   SyncWaitCount:        0
 QueuedSyncFuncs:        0
     UserAPCList:        0
     KernAPCList:        0
  pPMPSPSelector: C1F964A2
     BlockedOnID:        0
    TraceRefData:        0
   TraceCallBack:        0
  TraceOutLastCS:     0000
TraceEventHandle:     0000
          K16PDB:     1D57
       DosPDBSeg:     194C
  ExceptionCount:       00
```

#.WS - displays the status of all ring 3 system critical sections.

```
Win16Mutex  level=1 hnd=0001C7A4 unowned
Krn32Mutex  level=2 hnd=BFFD74F0 unowned
crstGHeap16 level=3 hnd=0001C7C8 unowned
crstLstMgr  level=3 hnd=BFFD7630 unowned
crstExcpt16 level=3 hnd=BFFD7520 unowned
crstSync    level=3 hnd=BFFD7550 unowned
```

#.WT [ppdb] or nothing for current process - Dump process handle table

```
Handle table of process ID 81681218 C:\WINDOWS\SYSTEM\KERNEL32.DLL
  Handle Object   Type
       1 81681218 Process (5)
       2 815839B0 Event (2)
       3 C1E8BED4 Mapped file (B)
       4 C1E8C09C Mapped file (B)
       5 C1E8C264 Mapped file (B)
       6 C1E8C42C Mapped file (B)
       7 815839DC Event (2)
       8 815853D8 Event (2)
       9 8158549C Event (2)
       A 815854E0 Thread (6)
       B 815858D4 Event (2)
       C C1F8DDFC Mapped file (B)
       D 81586438 Event (2)
       E 815870F8 Event (2)
       F 81588B34 Event (2)
      10 81588B90 Event (2)
      11 81588C38 Event (2)
      12 81588CC4 Event (2)
      13 81588D50 Event (2)
      14 81588DDC Event (2)
      15 81588E68 Semaphore (1)
      16 81588E94 Event (2)
      17 815898F0 Device IOCtl (D)
      18 81589968 Device IOCtl (D)
      19 8158A4F4 Device IOCtl (D)
      1B 8158B088 Event (2)
      1D 8158A57C Event (2)
      1E 8158A784 Event (2)
      1F 8158AC70 Process (5)
      20 8158F628 Device IOCtl (D)
```

.WMP [Process ID]

Dumps a list of modules (DLLs and EXE) visible in a process context. If no ppdb is passed in, dumps the modules for the current process. This is useful when you break into the debugger in some random place in 32-bit code and you say, 'where am I'? Just type .wmp and you will see a list of dlls in this process. Find the tow numbers bracketing where you are and the dll with the lower of the two addresses is the one you're in. Alternatively, if *you break in with <control>C and want to know where you're at in a certain process, type .w, then locate the thread you're interested in, type .r [Thread Number] to switch to this context. Then type .swp [Process ID of this thread] to list the modules owned by this process and bracket the given EIP. No symbols necessary for any of this.*

#.WMP [ppdb] - Dump module table for process (no ppdb dumps current process)

```
IMTE  3 81585248  4[1] BFC00000 C:\WINDOWS\SYSTEM\USER32.DLL
IMTE  1 81583C74  4[3] BFF20000 C:\WINDOWS\SYSTEM\GDI32.DLL
IMTE  2 81583FF0  4[1] BFED0000 C:\WINDOWS\SYSTEM\ADVAPI32.DLL
IMTE  0 81582248  5[4] BFF70000 C:\WINDOWS\SYSTEM\KERNEL32.DLL
```

#.Y <expr> ---- Displays a CONFIGMG structure

```
CM (space for help)?
Welcome to Configuration Manager's debugger
a Arbitrators    Show the list of arbitrators
b Block queue    Prevent processing of events
c Query remove   Query removal at happy time
d remove         Removal at happy time
e toggle Echo    Set the echo to a specified level
f show Free      Show free resources
g enumerate      Enumerate a devnode
h Reenable Appy  Reenable checking of query remove
k Show Log       Show procedural logs
l show List      Show devnode list
m toggle [more]  Toggle display stopping
n toggle name    Toggle the display of devnode name
p Problem        List devnodes with problems
q Quit           Quit the debugger
r show Range     Show a Range
s Show tree      Show the hardware tree
t toggle time    Toggle the display of time
u Unblock queue  Restart processing of events
v View status    View the global status of CONFIGMG
y force smthng   Call some random HWProfile API
z Allow DLL call Allow the DLLs to be called
```

CM (space for help)? S Show tree

```
Subtree starting at @C177AAA4
HTREE\ROOT\0   -HTREE\RESERVED\-NULL
               \ROOT\*PNP0000\0-NULL
               \ROOT\*PNP0200\0-NULL
               \ROOT\*PNP0B00\0-NULL
               \ROOT\*PNP0100\0-NULL
               \ROOT\*PNP0800\0-NULL
               \ROOT\*PNP0C04\0-NULL
               \ROOT\*PNP0C01\0-NULL
               \ROOT\*PNP0303\0-NULL
               \ROOT\*PNP812D\0-NETWORK\MSTCP\0-NETWORK\VREDIR\-NETWORK\VSERVER-NULL
                                               \NETWORK\VSERVER-NULL
                               \NETWORK\NWLINK\-NETWORK\VSERVER-NULL
                                               \NETWORK\VREDIR\-NETWORK\VSERVER-NULL
                                               \NETWORK\NWREDIR-NULL
                               \NETWORK\NETBEUI-NETWORK\VSERVER-NULL
                                               \NETWORK\VREDIR\-NETWORK\VSERVER-NULL
               \ROOT\*PNP0A03\0-PCI\VEN_8086&DE-NULL
                               \PCI\VEN_1002&DE-MONITOR\DEFAULT-NULL
                               \PCI\VEN_8086&DE-EISA\*PNP0A00\0-ISAPNP\READDATA-NULL
                               \PCI\VEN_1000&DE-SCSI\SONY____HS-NULL
                               \PCI\VEN_1000&DE-SCSI\DEC_____DS-NULL
               \ROOT\*PNP0F0E\0-NULL
               \ROOT\*PNP0700\0-FLOP\GENERIC_NE-NULL
                               \FLOP\GENERIC_NE-NULL
               \ROOT\*PNP0500\0-NULL
               \ROOT\*PNP0500\0-NULL
               \ROOT\*PNP0400\0-NULL
               \ROOT\NET\0000  -NETWORK\MSTCP\0-NETWORK\VREDIR\-NETWORK\VSERVER-NULL
                                               \NETWORK\VSERVER-NULL
                               \NETWORK\NWLINK\-NETWORK\VSERVER-NULL
                                               \NETWORK\VREDIR\-NETWORK\VSERVER-NULL
                                               \NETWORK\NWREDIR-NULL
                               \NETWORK\NETBEUI-NETWORK\VSERVER-NULL
                                               \NETWORK\VREDIR\-NETWORK\VSERVER-NULL
               \ROOT\PRINTER\00-NULL
               \ROOT\PRINTER\00-NULL
```

```
CM (space for help)? L show List
  Walk(00000002,HTREE\ROOT\0,00000000,00000000,C006A157);
DN=C179365C, PB=00000000, ID=HTREE\RESERVED\0
DN=C1793760, PB=00000000, ID=ROOT\*PNP0000\0000
DN=C1793A5C, PB=00000000, ID=ROOT\*PNP0200\0000
<<data omitted>>
DN=C17D928C, PB=00000000, ID=FLOP\GENERIC_NEC__FLOPPY_DISK_\ROOT&*PNP0700&000000
DN=C17D9344, PB=00000000, ID=FLOP\GENERIC_NEC__FLOPPY_DISK_\ROOT&*PNP0700&000010
<<data omitted>>
DN=C1796014, PB=00000000, ID=ROOT\PRINTER\0001
DN=C177AAA4, PB=00000000, ID=HTREE\ROOT\0
  return(00000001);

CM (space for help)? G enumerate
DevNode? C17D9344
  AssertDevNode(FLOP\GENERIC_NEC__FLOPPY_DISK_\ROOT&*PNP0700&000010);
  return(00000001);
Scheduled reenumeration of FLOP\GENERIC_NEC__FLOPPY_DISK_\ROOT&*PNP0700&000010
CONFIGMG_Reenumerate_DevNode(FLOP\GENERIC_NEC__FLOPPY_DISK_\ROOT&*PNP0700&000010,00000000);
   WillNeedReenumerateDevNode(FLOP\GENERIC_NEC__FLOPPY_DISK_\ROOT&*PNP0700&000010);
    AsyncAppyTime(C0256399,FLOP\GENERIC_NEC__FLOPPY_DISK_\ROOT&*PNP0700&000010,DEADBEEF,DEADBEEF);
     AllocateAppyTime(C0256399,FLOP\GENERIC_NEC__FLOPPY_DISK_\ROOT&*PNP0700&000010);
     return(C163AAF8);
    return;
   return;
  return(CR_SUCCESS);
```

VxD commands

VxD commands are debugging commands provided by the actual VxD binaries. These commands are subject to change without notice, and often are available only in the debug version of the component. Note that some of these commands begin with a double dot.

```
#.DOSMGR
Select new DOSMGR function:
  [1] Display SFTs
  [2] Display DPBs
  [3] Display MCBs
  [4] Display CDSs
  [5] Display PDBs
  [6] Display DEVs
  [7] Display BDSs
  [8] Display INT 21h status
  [9] Display other (old) DOSMGR options

? [1] Display SFTs
00C900CC:
    00: sf_ref_count      0008
    02: sf_mode           0002
    04: sf_attr           00
    05: sf_flags          00C0
    07: sf_devptr         00700028
    0B: sf_firclus        0028
    0D: sf_time           7943
    0F: sf_date           1F03
    11: sf_size           00000000
    15: sf_position       00000000
    19: sf_cluspos        0000
    1B: sf_dirsec         00000000
    1F: sf_dirpos         00
    20: sf_name           "AUX         "
    2B: sf_chain          00000000
    2F: sf_uid            0000
    31: sf_pid            8635
    33: sf_mft            0000
    35: sf_lstclus        0000
    37: sf_ifs_hdr        00000000
Press a key to continue...

? [2] Display DPBs
00C91346:
    00: dpb_drive         00
    01: dpb_unit          00
    02: dpb_sector_size   0200
    04: dpb_cluster_mask  FE
    05: dpb_cluster_shift 00
    06: dpb_first_fat     0001
    08: dpb_fat_count     02
    09: dpb_root_entries  0040
    0B: dpb_first_sector  0009
    0D: dpb_max_cluster   0160
    0F: dpb_fat_size      0002
    11: dpb_dir_sector    0005
    13: dpb_driver_addr   0070005E
    17: dpb_media         00
    18: dpb_first_access  FF
    19: dpb_next_dpb      00C91367
    1D: dpb_next_free     0000
    1F: dpb_free_cnt      FFFF
Press a key to continue...
00C91367:
    00: dpb_drive         01
    01: dpb_unit          01
    02: dpb_sector_size   0200
    04: dpb_cluster_mask  FE
    05: dpb_cluster_shift 00
    06: dpb_first_fat     0001
    08: dpb_fat_count     02
    09: dpb_root_entries  0040
    0B: dpb_first_sector  0009
    0D: dpb_max_cluster   0160
    0F: dpb_fat_size      0002
    11: dpb_dir_sector    0005
    13: dpb_driver_addr   0070005E
    17: dpb_media         00
    18: dpb_first_access  FF
    19: dpb_next_dpb      00C91388
    1D: dpb_next_free     0000
    1F: dpb_free_cnt      FFFF
Press a key to continue...
```

? [3] Display MCBs

```
  0206:   sig=M owner=0008 size=0405 name=MS-DOS
  0207:   sig=D owner=0208 size=0044 name=HIMEM
  024C:   sig=D owner=024D size=0098 name=DBLBUFF
  02E5:   sig=D owner=02E6 size=00B3 name=IFSHLP
  0399:   sig=D owner=039A size=0033 name=SETVER
  03CD:   sig=I owner=03CE size=0022 name=
  03F0:   sig=I owner=03F1 size=0019 name=
  040A:   sig=F owner=040B size=005D name=
  0468:   sig=X owner=0469 size=0010 namev~h▯▯D
  0479:   sig=B owner=047A size=0020 name=]CUl8
  049A:   sig=L owner=049B size=00B0 name=▯>8▯
  060C:   sig=M owner=0008 size=0004 name=MS-DOS
  0611:   sig=M owner=061E size=000B name=<Env>
  061D:   sig=M owner=061E size=00D0 name=WIN
  06EE:   sig=M owner=06F2 size=0002 name=
  06F1:   sig=M owner=06F2 size=0452 name=vmm32
  0B44:   sig=M owner=0B49 size=0003 name=
  0B48:   sig=Z owner=0B49 size=92B7 name=KRNL386
  FFFF:0020:    owner=FFFF size=2F10
  FFFF:2F40:    owner=FF03 size=8600
  FFFF:B550:    owner=0001 size=0AE0
  FFFF:C040:    owner=FFFB size=0280
  FFFF:C2D0:    owner=0000 size=0D30
  FFFF:D010:    owner=FFFA size=2FD0
```

? [4] Display CDSs

```
049B0000:
    00: curdir_text       "A:\ "
    43: curdir_flags      4000
    45: curdir_devptr     00C91346
    49: curdir_id         FFFFFFFF
    4D: curdir_user_word  6EB2
    4F: curdir_end        0002
    51: curdir_type       00
    52: curdir_ifs_hdr    02E60AC8
    56: curdir_fsda       0000
Press a key to continue...
049B00B0:
    00: curdir_text       "C:\WINDOWS\DESKTOP "
    43: curdir_flags      4000
    45: curdir_devptr     00C91388
    49: curdir_id         FFFFFFFF
    4D: curdir_user_word  6EB2
    4F: curdir_end        0002
    51: curdir_type       00
    52: curdir_ifs_hdr    02E60AC8
    56: curdir_fsda       0000
```

? [5] Display PDBs

```
00000B48:
    01: pdb_owner         0B49
    03: pdb_size          92B7
    08: pdb_name          "KRNL386 "
    10: pdb_exit_call     20CD
    12: pdb_block_len     9E00
    1A: pdb_exit          FC8B184F
    1E: pdb_ctrl_c        FC8418BF
    22: pdb_fatal_abort   061E080B
    26: pdb_parent_pid    06F2
    28: pdb_jfn_table     07 01 01 00 02 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    3C: pdb_environ       017F
    3E: pdb_user_stack    0B590228
    42: pdb_jfn_length    0080
    44: pdb_jfn_pointer   25120000
    48: pdb_next_pdb      FFFF
    4C: pdb_intercon      00
    4D: pdb_append        00
    50: pdb_version       0007
Press a key to continue...
```

? [6] Display DEVs

```
00C90048:
    00: SDEVNEXT          039A0000
    04: SDEVATT           8004
    06: SDEVSTRAT         0DC7
    08: SDEVINT           0DCD
    0A: SDEVNAME          "NUL     "
Press a key to continue...
02E60000:
    00: SDEVNEXT          024D0000
    04: SDEVATT           D000
    06: SDEVSTRAT         040B
    08: SDEVINT           03A8
    0A: SDEVNAME          "IFS$HLP$ "
Press a key to continue...
024D0000:
    00: SDEVNEXT          02080000
    04: SDEVATT           C840
    06: SDEVSTRAT         0016
    08: SDEVINT           0021
    0A: SDEVNAME          "DblBuff$ "
Press a key to continue...
```

? [7] Display BDSs

```
03F10000:
    00: bds_link          03F10064
    04: bds_drivenum      00
    05: bds_drivelet      00
    06: bds_BPB.BytesSec  0200
    08: bds_BPB.SecPerClu FF
    09: bds_BPB.ResSec    0001
    0B: bds_BPB.NumFATs   02
    0C: bds_BPB.RootEnts  0040
    0E: bds_BPB.TotalSec  0168
    10: bds_BPB.MediaDesc 00
    11: bds_BPB.SecPerFAT 0002
    13: bds_BPB.SecPerTrk 0009
    15: bds_BPB.Heads     0001
    17: bds_BPB.HiddenSec 00000000
    1B: bds_BPB.BigTotSec 00000000
    1F: bds_fatsiz        00
    20: bds_opcnt         0000
    22: bds_formfactor    07
    23: bds_flags         0022
    25: bds_ccyln         0050
    27: bds_RBP.BytesSec  0200
    29: bds_RBP.SecPerClu 01
    2A: bds_RBP.ResSec    0001
    2C: bds_RBP.NumFATs   02
    2D: bds_RBP.RootEnts  00E0
    2F: bds_RBP.TotalSec  0B40
    31: bds_RBP.MediaDesc F0
    32: bds_RBP.SecPerFAT 0009
    34: bds_RBP.SecPerTrk 0012
    36: bds_RBP.Heads     0002
    38: bds_RBP.HiddenSec 00000000
    3C: bds_RBP.BigTotSec 00000000
    40: bds_RBP.Reserved  00 00 00 00 00 00
    46: bds_track         FF
    47: bds_tim_lo        FFFF
    49: bds_tim_hi        FFFF
    4B: bds_volid         "NO NAME    "
    57: bds_vol_serial    00000000
    5B: bds_filesys_id    "FAT12   "
```

? [8] Display INT 21h status

```
  InDOS: 00
    C91308: DrvMap: 01 01 07 07 01 00 00 00 00 00 00 00 00
                    00 00 00 00 00 00 00 00 00 00 00 00 00
```

#..ifsmgr

```
   [a]   - drive info
   [b]   - block info
   [c]   - shell resource info
   [d]   - open SFT files
   [e]   - open extended files
   [f]   - PerVM data
   [g]   - Heap Block info
   [h]   - Heap Allocation info
   [i]   - Trace Log
   [j]   - Set Log Mask
   [k]   - Display ifsreq
   [l]   - open files on volume
   [m]   - Display fhandle
   [n]   - Toggle volume lock write state trapping
   [o]   - Display threads in ifsmgr
   [ESC] to quit
```

? a

```
A: - FlatFat
    ShRes: Drive A:  (C1F8E044)   at C1F8DF94
sr_sig='Sr'              sr_serial=00                sr_idx=01
sr_next=C1F85274         sr_rh=C1F8E044              sr_func=C00AA0D4
sr_inUse=00000001        sr_uword=0000               sr_HndCnt=0000
sr_UNCCnt=00             sr_DrvCnt=01                sr_rtype=01
sr_flags=10              sr_ProID=00000000           sr_VolInfo=C1F8E004
sr_fhandleHead=00000000  sr_LockPid=00000000         sr_LockSavFunc=00000000
sr_LockType=00           sr_LockFlags=00             sr_PhysUnit=FF
sr_LockWaitCnt=00        sr_LockOwner=00000000       sr_LockReadCnt=0000
sr_LockWriteCnt=0000     sr_flags2=00                sr_pnv=00000000

C: - FlatFat
    ShRes: Drive C:  (C1F852E4)   at C1F85274
sr_sig='Sr'              sr_serial=00                sr_idx=00
sr_next=00000000         sr_rh=C1F852E4              sr_func=C008BF68
sr_inUse=00000043        sr_uword=0002               sr_HndCnt=0042
sr_UNCCnt=00             sr_DrvCnt=01                sr_rtype=01
sr_flags=10              sr_ProID=00000002           sr_VolInfo=C1F63380
sr_fhandleHead=C1F9C524  sr_LockPid=00000000         sr_LockSavFunc=00000000
sr_LockType=00           sr_LockFlags=02             sr_PhysUnit=FF
sr_LockWaitCnt=00        sr_LockOwner=00000000       sr_LockReadCnt=0000
sr_LockWriteCnt=0000     sr_flags2=00                sr_pnv=C1F99A78

Drive D is unmounting
Drive E is unmounting

M: - Network
    ShRes:    at C1F98340
sr_sig='Sr'              sr_serial=06                sr_idx=04
sr_next=00000000         sr_rh=C1541E14              sr_func=C00CB618
sr_inUse=00000001        sr_uword=0000               sr_HndCnt=0000
sr_UNCCnt=00             sr_DrvCnt=01                sr_rtype=01
sr_flags=08              sr_ProID=0000000B           sr_VolInfo=00000000
sr_fhandleHead=00000000  sr_LockPid=00000000         sr_LockSavFunc=00000000
sr_LockType=00           sr_LockFlags=00             sr_PhysUnit=00
sr_LockWaitCnt=00        sr_LockOwner=00000000       sr_LockReadCnt=0000
sr_LockWriteCnt=0000     sr_flags2=00                sr_pnv=00000000
```

#.NETBEUI

```
--- N E T B E U I   D E B U G   C O M M A N D S ---
[L]  -  display Log of recently received frames
[I]  -  display status of I-frame descriptors
[Q]  -  Quit Netbeui debugger (or G)
```

Enter selection or [ESC] to quit: I

```
00000032 free I-frame descriptors of 00000032
There are 00000000 dscs on LTE queues, 00000032 free of 00000032
There are 00000032 dscs on I_DSCFree (should be 00000032)

Enter selection or [ESC] to quit:
DLC_UI  |NAME_Q  CHRISG          * to MMPRODUCTS1
DLC_UI  |NAME_Q  ************ª*Jã} to red-29-msg     s
DLC_UI  |NAME_Q  BRUCEB1         * to JONL_SR
DLC_UI  |NAME_Q  ************€_ø*2 to red-40-msg.srv**
DLC_UI  |DGRAM   BRYANTH755        to SYS-MSDOS-WIN  *
DLC_UI  |DGRAM   TAKUA5            to SYS-MSDOS-WIN  *
DLC_UI  |DGRAM   T-JASONT       * to SYS-WIN4       *
DLC_UI  |NAME_Q  CHRISG         * to MMPRODUCTS1
DLC_UI  |DGRAM   CHRISFRA2      * to SYS-WIN4       *
DLC_UI  |NAME_Q  ***********ª*Jã} to red-29-msg     s
DLC_UI  |DGRAM   V-GORDF        * to SYS-WIN4       *
DLC_UI  |DGRAM   SEAWOLF          to NTWKSTA        *
DLC_UI  |DGRAM   GREY           * to SYS-WIN4       *
DLC_UI  |NAME_Q  ***********€_ø*2 to red-40-msg.srv**
DLC_UI  |DGRAM   T-TIANM        * to PSG-GDI        *
```

#.NWREDIR

```
[NoBall]  Netware Compatible Client
[NoBall]   P    - Display Print Capture Flags
[NoBall]   A    - Toggle API Tracing [OFF]
[NoBall]   L    - Toggle LFN Support [1]
[NoBall]   T0-3 - Toggle Full trace [0]
[NoBall]   R    - Toggle Read Caching [ON]
[NoBall]   B    - Toggle Burst Support [ON]
[NoBall]   S    - Structure Dump
[NoBall]   D    - Dump trace log
[NoBall]   C    - Dump connection table
[NoBall]   N    - Dump net resources
[NoBall]   Esc  - Return to Debugger
co   *** Connection 0xC1F651D8 CHICOPRT *** FAST
co   Username PERCYT
co         co_current_iob 0x00000000           co_queue_count 0x00000000
co      co_flush_asap_cnt 0x00000000         co_nsec_per_byte 0x00000ABE
co       co_write_timeout 0x00000000            co_bwrite_cnt 0x00000000
co            co_exit_cnt 0x00000000         co_TimeoutHandle 0x00000000
co               co_InUse 0x000000FF                 co_Block 0x00000000
co             co_Waiting 0x00000000                co_Tickle 0x00000001
co        co_ConnectionId 0x0009E7A0            co_SendPacket 0xC1F98F3C
co          co_RecvPacket 0xC1F99264                   co_bcb 0xC1F994F8
co          co_PacketSize 0x000005B8            co_BufferSize 0x000005A4
co           co_OpenFiles 0x00000000             co_TickCount 0x00000001
co            co_MaxRetry 0x00000014        co_MessageWaiting 0x00000000
co               co_Index 0x00000001          co_MaxFragments 0x00000011
co         co_ExtraTimeMS 0x00000000             co_TimeOutMS 0x0000006E
co        co_MaxTimeOutMS 0x00000339            co_SequenceNo 0x00000016
co        co_ConnectionLo 0x00000027          co_ConnectionHi 0x00000000
co        co_MajorVersion 0x00000001          co_MinorVersion 0x0000000C
co           co_Signature 0x4E4E4F43            co_nsec_delay 0x00353518
co           bcb_SendSize 0x000037B0             bcb_RecvSize 0x000053A4
co        bcb_SendGapTime 0x00000FBA          bcb_RecvGapTime 0x00000FBA
co bcb_LastBadSendGapTime 0x00000000   bcb_LastBadRecvGapTime 0x00000000
co              bcb_Flags 0x00000000         bcb_NoErrorSlope 0x00000010
co    bcb_SmallErrorSlope 0x00000004      bcb_LargeErrorSlope 0x00000002
** No pending IOBlks **

[NoBall]  Netware Compatible Client
[NoBall]   P    - Display Print Capture Flags
[NoBall]   A    - Toggle API Tracing [OFF]
[NoBall]   L    - Toggle LFN Support [1]
[NoBall]   T0-3 - Toggle Full trace [0]
[NoBall]   R    - Toggle Read Caching [ON]
[NoBall]   B    - Toggle Burst Support [ON]
[NoBall]   S    - Structure Dump
[NoBall]   D    - Dump trace log
[NoBall]   C    - Dump connection table
[NoBall]   N    - Dump net resources
[NoBall]   Esc  - Return to Debugger
```

#.SHELL

```
Shell VxD -- SH_CB_Offset = 000005EC
C1D20154 hwnd 0000, Error code 00 Title: Windows
```

Wshell: [G]eneral, [V]M, [T]race, [A]ppy, [B]roadcast, [P]roperties: G

```
C1D20154 hwnd 0000, Error code 00 Title: Windows
Wshell: [G]eneral, [V]M, [T]race, [A]ppy, [B]roadcast, [P]roperties:
WshCbOffset = 000014C4
User Busy FLAGS -
SYS_VM_BPriAdjusted = 00      , SYS_VM_FPriAdjusted = 00
SYS_VM_TrueBpri     = 00000000, SYS_VM_TrueFpri     = 00000000
VDD_Grab_Addr = C02373B0:VDD(07) + 4E5
No global boost active
```

```
#.V86MMGR
Select desired V86MMGR component:
   [0]   - General info
   [1]   - Memory scan info
   [2]   - EMM driver info
   [3]   - XMS driver info
   [ESC] - Exit V86MMGR debug querry
 ? [0]   - General info
*******************************************************************************
           G E N E R A L   V 8 6 M M G R   D R I V E R   S T A T U S

   Global V86MMGR Flags = LOCA20

[ESC] to exit, any other key for VM dump
VM C1D20154
    Local VM pages array is set
    VM Base Memory size is 00000094
    Lead Handle 00000000, Main Handle 80004000, Tail Handle 00000000
```

```
#.VCACHE
[A] - Dump client information.
[B] - Change the debug flags.
[X] - Exit menu.
? [A] - Dump client information.
ID Min      Extra    Discard
64 1        FFFFFFFF C00FB6E8:VCacheT(01)
65 4        54D      VFAT_Discard
66 0        0        C003F594:NWREDIR(01) + EA0
67 0        C        C00C461C:VREDIR(01) + 738
```

```
#.VCD
COM01:
  Base    Address   Offs     Value  Field name
C1E7EF44  C1E7EF44  0000  000006D8  VCD_CB_Offset
C1E7EF44  C1E7EF48  0004        01  VCD_Number
C1E7EF44  C1E7EF49  0005        04  VCD_IRQN
C1E7EF44  C1E7EF4C  0008  00000000  VCD_IRQ_Desc
C1E7EF44  C1E7EF4A  0006      0080  VCD_Flags
C1E7EF44  C1E7EF5C  0018  000003F8  VCD_IObase
C1E7EF44  C1E7EF60  001C  00000000  VCD_Owner
C1E7EF44  C1E7EF64  0020  00000000  VCD_IRQ_Handle
C1E7EF44  C1E7EF78  0034        30  VCD_Def_BAUD_LSB
C1E7EF44  C1E7EF79  0035        00  VCD_Def_BAUD_MSB
C1E7EF44  C1E7EF7A  0036        00  VCD_Def_IER
C1E7EF44  C1E7EF7B  0037        03  VCD_Def_LCR
C1E7EF44  C1E7EF7C  0038        00  VCD_Def_MCR
C1E7EF44  C1E7EF7D  0039        60  VCD_Def_LSR
C1E7EF44  C1E7EF7E  003A        00  VCD_Def_MSR

STATE OF VM C1D20154:
  Base    Address   Offs     Value  Field name
C1D2082C  C1D2082C  0000        60  VCD_BAUD_LSB
C1D2082C  C1D2082D  0001        00  VCD_BAUD_MSB
C1D2082C  C1D2082E  0002        00  VCD_IER
C1D2082C  C1D2082F  0003        02  VCD_LCR
C1D2082C  C1D20830  0004        00  VCD_MCR
C1D2082C  C1D20831  0005        00  VCD_Read_Stat
C1D2082C  C1D20832  0006      0000  VCD_CB_Flags

COM02:
  Base    Address   Offs     Value  Field name
C1E934E0  C1E934E0  0000  00000700  VCD_CB_Offset
C1E934E0  C1E934E4  0004        02  VCD_Number
C1E934E0  C1E934E5  0005        03  VCD_IRQN
C1E934E0  C1E934E8  0008  00000000  VCD_IRQ_Desc
C1E934E0  C1E934E6  0006      0080  VCD_Flags
C1E934E0  C1E934F8  0018  000002F8  VCD_IObase
C1E934E0  C1E934FC  001C  00000000  VCD_Owner
C1E934E0  C1E93500  0020  00000000  VCD_IRQ_Handle
C1E934E0  C1E93514  0034        06  VCD_Def_BAUD_LSB
C1E934E0  C1E93515  0035        00  VCD_Def_BAUD_MSB
C1E934E0  C1E93516  0036        00  VCD_Def_IER
C1E934E0  C1E93517  0037        03  VCD_Def_LCR
C1E934E0  C1E93518  0038        00  VCD_Def_MCR
```

```
C1E934E0  C1E93519  0039        60  VCD_Def_LSR
C1E934E0  C1E9351A  003A        B0  VCD_Def_MSR

STATE OF VM C1D20154:
  Base    Address   Offs     Value  Field name
C1D20854  C1D20854  0000        06  VCD_BAUD_LSB
C1D20854  C1D20855  0001        00  VCD_BAUD_MSB
C1D20854  C1D20856  0002        00  VCD_IER
C1D20854  C1D20857  0003        03  VCD_LCR
C1D20854  C1D20858  0004        00  VCD_MCR
C1D20854  C1D20859  0005        00  VCD_Read_Stat
C1D20854  C1D2085A  0006      0000  VCD_CB_Flags
```

#.VCOMM

```
VCOMM services:
[1] Dump the list of supported ports
[2] Look at profiles
[3] Reset Profiles
Enter selection or press [ESC] to exit:
```

#.VDMAD

```
VDMAD state
 Handle      Size     Physical    Linear      Time     Owner VM   State
C00B209C   (Unallocated)
C00B20B4   (Unallocated)
C00B20CC   (Unallocated)
C00B20E4   (Unallocated)
C00B20FC   (Unallocated)
C00B2114   (Unallocated)
C00B212C   (Unallocated)
C00B2144   (Unallocated)

CB offset          0007FE88
VPICD HW INT notification not hooked

ESC to quit, or any char to see controller states:
Controller          1                 2
status             00                00
mask               0F                0F
request            00                00
                adr     cnt   mod em      adr     cnt   mod em
channel 00   00000000 00000000 40 00  00000000 00000000 40 00
channel 01   00000000 00000000 40 00  00000000 00000000 40 40
channel 02   00000000 00000000 40 00  00000000 00000000 40 40
channel 03   00000000 00000000 40 00  00000000 00000000 40 40

ESC to quit, or any char to see channel states:
DMA channel #00  max=0000Kb  callback = VDMAD_Call_Def
DMA channel #01  max=0000Kb  callback = VDMAD_Call_Def
DMA channel #02  max=0000Kb  callback = C0294D4C:VFBACKUP(04) + 704
DMA channel #03  max=0000Kb  callback = VDMAD_Call_Def
DMA channel #04  max=0000Kb  callback = none
DMA channel #05  max=0000Kb  callback = VDMAD_Call_Def
DMA channel #06  max=0000Kb  callback = VDMAD_Call_Def
DMA channel #07  max=0000Kb  callback = VDMAD_Call_Def

ESC to quit, or any char to see CurVM states:
Virtual DMA State for CurVM C1D20154
flipflop:  ctrl 1 00   ctrl 2 00
```

#.VDD

```
VDD video state dump for all Virtual Machines:
Vid_Flags         = fVid_MsgInit fVid_SysVMI fVid_VMsSuspended fVid_PlanarInit f
Vid_DspDrvr_Init
VT_Flags          = fVT_NoSaveResSysVM fVT_Mono fVT_MonoPorts
Vid_CB_Off        = 0007F11C
Vid_Focus_VM      = C4920154
Vid_Msg_VM        = 00000000
Vid_CrtC_VM       = C4920154
Vid_MemC_VM       = C4920154
Vid_Msg_Pseudo_VM = C008B104

Select Option
  1 - display VM register states
  2 - display VM memory usage
  3 - dump video page info
  4 - display msg mode register state
  5 - display planar mode register state
  6 - read DAC
  7 - display VM DAC states
  8 - enable Queue_Outs
  9 - enable MemC debug event
```

#.VFAT

```
[A] - Toggle empty field printing.
[B] - Dump the OFT.
[C] - Dump the log.
[D] - Change the log mask.
[E] - Dump the volume list.
[F] - Dump the directory cache.
[G] - Dump buffer header.
[H] - Dump buffer chain.
[I] - Dump IOREQ.
[J] - Dump VFAT SFT.
[K] - Flush the log.
[L] - Dump threads in VFAT.
[X] - Exit menu.
? [E] - Dump the volume list.
VolTab Address = C1F852E4
Vol_Sig = 'VOLT'          Vol_Label = '           ' Vol_BdLlen = 00
Vol_BootID = 1ED44C40     Vol_BufSkew = 00000001    Vol_ChiSerial = 00000000
Vol_ClusBmask = 00003FFF  Vol_ClusBshift = 0E       Vol_ClusterMask = 1F
Vol_ClusterShift = 05     Vol_DirSector = 00000191  Vol_DirtCnt = 00000000
Vol_Drive = 02            Vol_DriveOrig = 02        Vol_FatCount = 02
Vol_FatEnd = 000000C9     Vol_FatSize = 000000C8    Vol_FirstFat = 00000001
Vol_FirstSector = 000001B1                          Vol_Flags = 0000400A
Vol_FlushHead = 00000000  Vol_FreeCnt = 000062B7    Vol_LastATime = 0005D9D4
Vol_MaxCluster = 0000C7FF Vol_MaxSec = 00190180     Vol_Media = F8
Vol_MsDataAge = 000007D0  Vol_MsIdleBuf = 000003E8  Vol_NameHashTable = C17DC770
Vol_NextFree = 000054DA   Vol_Offset = 00000000     Vol_RaMask = 0000F000
Vol_RasCnt = 01           Vol_RdCnt = 00000000      Vol_RdFlush = 00000000
Vol_RootEntries = 00000200                          Vol_RootSum = 00000000
Vol_SectorMask = 000001FF Vol_SectorShift = 09      Vol_SectorSize = 00000200
Vol_Sflag = 02            Vol_SPB = 08              Vol_SPBMask = 07
Vol_TagActive = 00000000  Vol_TagBitNext = 01       Vol_VolIdle = 000001F4
Vol_Vrp = C1F63380        Vol_WrCnt = 00000000      Vol_WrHead = 00000000
Vol_WrHint = 00000000     Vol_WrPri = 01
```

#.VFLATD

```
00000000   VFLATD_Fault_Count
00000000   VFLATD_Video_Base
00000000   VFLATD_Video_Size
00000000   VFLATD_Video_Flags
00000000   VFLATD_Video_Sel
00000000   VFLATD_BankSize
00000000   VFLATD_BankAddress
00000000   VFLATD_Linear_Access_Count
hit any key for VflatD event log
```

#.VKD

```
Select desired VKD debug function:
   [0]   - General info
   [1]   - Hot Key info
   [2]   - Per VM info
   [3]   - Set VKD queue_outs
   [ESC] - Exit VKD debug querry
 ? [0]   - General info
VKD STATUS
focus:  C1D20154    (CB offset = 0007F768)
global shift state: 0000
VKD flags: 00000010
global keybuf =
recent global keybuf = 9C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0
0 00 00 00 00 00 00 00 00 00 00 2A 14 94 23 AA A3 16 96 31 20 B1 A0 12 92 13 93
2A AA 30 B0 1E 9E 15 95 0F 8F 0F 8F 0F 8F 0F 8F 19 99 12 92 13 93 2E AE 15 95 14
94 1C
msg mode buffer owner: 00000000
```

#.VNETBIOS

```
VM C1D20154 has no more hook control blocks allocated
```

#.VREDIR

```
   [a]   - session info
   [b]   - nib info
   [c]   - Active NCBs
   [d]   - All NCBs
   [e]   - net info
   [f]   - beacons
   [g]   - Enum Info
   [h]   - MsgBuf
   [i]   - User Info
   [ESC] to quit
 ? [a]   - session info
Server: HITME  uniname :\HITME: (C159A454)
  s_options=00 USec Encr
  s_state=80 Disconn
  s_active=0000, s_nibcount=0000, s_maxreq=0001
s_activeUsers=0001
(0) = 65535 SMBUID_NOT_SETUP
(1) = 8194
(2) = 65535 SMBUID_NOT_SETUP
  s_net=0, s_lsn=0
  Resource: :WEEKLY:  (C1541E14)
    r_type=01, r_flags=82 DisOK Disconn
```

#.VPICD

```
***VPICD DEBUG INFORMATON ***
[1] Global PIC information
[2] Per-VM PIC information
[3] IRQ handler information
>[1] Global PIC information
----- Int NUMBER -----  08 09 0A 0B 0C 0D 0E 0F 70 71 72 73 74 75 76 77
----- IRQ NUMBER -----  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
Physically in service:   .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
Physically masked:       .  .  .  Y  Y  Y  .  Y  .  .  .  .  .  .  Y  .
Physically requested:    Y  .  Y  .  .  .  .  .  .  .  Y  .  .  .  .  .
Virtualized:             Y  Y  Y  .  .  .  Y  .  .  .  Y  Y  Y  Y  .  Y
Globally owned:          Y  Y  Y  .  .  .  Y  .  Y  Y  .  .  .  Y  .  .
Shared:                  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
Alloc'd by arbitrator:   Y  Y  Y  Y  Y  .  Y  .  Y  Y  Y  Y  Y  Y  .  Y
        Preallocated:    .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
        Shareable:       .  .  .  Y  Y  .  .  .  .  Y  .  Y  .  .  .  Y
        Reserved:        Y  Y  Y  Y  Y  .  Y  Y  Y  Y  Y  Y  Y  Y  .  Y
Masked at boot time:     .  .  .  Y  Y  Y  .  Y  .  .  Y  Y  Y  .  Y  Y
Auto mask at inst swap:  .  .  .  .  .  .  .  .  .  .  .  Y  .  .  .  Y

>[3] IRQ handler information
IRQ 00: Hw_Int_Proc=VTD_INT, Refdata=C00AEA34
        Virt_Int_Proc=VTD_Virt_Int, EOI_Proc=VTD_EOI
        Mask_Change_Proc=0028:00000000, IRET_Proc=VTD_IRET

IRQ 01: Hw_Int_Proc=VKD_Int_09, Refdata=00000000
        Virt_Int_Proc=VKD_Virt_Int, EOI_Proc=VKD_EOI
        Mask_Change_Proc=0028:00000000, IRET_Proc=VKD_IRET

IRQ 02: Hw_Int_Proc=VPICD_Slave_Error, Refdata=00000000
        Virt_Int_Proc=0028:00000000, EOI_Proc=Default_Cascade_EOI
        Mask_Change_Proc=VPICD_Slave_Mask_Change_Proc, IRET_Proc=VPICD_Slave_Error

IRQ 03: Hw_Int_Proc=Default_INT, Refdata=00000000
        Virt_Int_Proc=0028:00000000, EOI_Proc=Default_EOI
        Mask_Change_Proc=Default_MASK, IRET_Proc=Default_Iret

IRQ 04: Hw_Int_Proc=Default_INT, Refdata=00000000
        Virt_Int_Proc=0028:00000000, EOI_Proc=Default_EOI
        Mask_Change_Proc=Default_MASK, IRET_Proc=Default_Iret

IRQ 05: Hw_Int_Proc=Default_INT, Refdata=00000000
        Virt_Int_Proc=0028:00000000, EOI_Proc=Default_EOI
        Mask_Change_Proc=Default_MASK, IRET_Proc=Default_Iret

IRQ 06: Hw_Int_Proc=Floppy_Hw_Int, Refdata=CCCCCCCC
        Virt_Int_Proc=0028:00000000, EOI_Proc=Floppy_Hw_EOI
        Mask_Change_Proc=0028:00000000, IRET_Proc=0028:00000000

IRQ 06: Hw_Int_Proc=Default_INT, Refdata=00000000
        Virt_Int_Proc=0028:00000000, EOI_Proc=Default_Shared_EOI
        Mask_Change_Proc=0028:00000000, IRET_Proc=Default_Shared_IRET

IRQ 07: Hw_Int_Proc=Default_INT, Refdata=00000000
        Virt_Int_Proc=0028:00000000, EOI_Proc=Default_EOI
        Mask_Change_Proc=Default_MASK, IRET_Proc=Default_Iret

IRQ 08: Hw_Int_Proc=Default_INT, Refdata=00000000
        Virt_Int_Proc=0028:00000000, EOI_Proc=Default_EOI
        Mask_Change_Proc=Default_MASK, IRET_Proc=Default_Iret

IRQ 09: Hw_Int_Proc=Def_Int_IRQ9_Owner_Detect, Refdata=00000000
        Virt_Int_Proc=0028:00000000, EOI_Proc=Default_IRQ9_EOI
        Mask_Change_Proc=Default_MASK, IRET_Proc=Default_Iret

IRQ 10: Hw_Int_Proc=C00D18D8:NDIS(01) + 5DD2, Refdata=C1E94698
        Virt_Int_Proc=0028:00000000, EOI_Proc=0028:00000000
        Mask_Change_Proc=0028:00000000, IRET_Proc=0028:00000000

IRQ 11: Hw_Int_Proc=_Port_ISR, Refdata=C1A21414
        Virt_Int_Proc=0028:00000000, EOI_Proc=0028:00000000
        Mask_Change_Proc=0028:00000000, IRET_Proc=0028:00000000

IRQ 11: Hw_Int_Proc=Default_PM_Share_Int, Refdata=00000000
        Virt_Int_Proc=0028:00000000, EOI_Proc=0028:00000000
        Mask_Change_Proc=0028:00000000, IRET_Proc=0028:00000000

IRQ 12: Hw_Int_Proc=VAD_INT, Refdata=00000000
        Virt_Int_Proc=0028:00000000, EOI_Proc=0028:00000000
        Mask_Change_Proc=VMD_Mask_Change_Proc, IRET_Proc=0028:00000000

IRQ 12: Hw_Int_Proc=Def_Int_Share_Error, Refdata=00000000
        Virt_Int_Proc=0028:00000000, EOI_Proc=0028:00000000
        Mask_Change_Proc=0028:00000000, IRET_Proc=0028:00000000

IRQ 13: Hw_Int_Proc=C010254C:PharLap(01) + FC, Refdata=CCCCCCCC
        Virt_Int_Proc=0028:00000000, EOI_Proc=0028:00000000
        Mask_Change_Proc=0028:00000000, IRET_Proc=0028:00000000

IRQ 13: Hw_Int_Proc=VMCPD_IRQ13, Refdata=CCCCCCCC
        Virt_Int_Proc=VMCPD_Virt_IRQ13, EOI_Proc=VMCPD_EOI
        Mask_Change_Proc=0028:00000000, IRET_Proc=0028:00000000

IRQ 13: Hw_Int_Proc=Default_INT, Refdata=00000000
        Virt_Int_Proc=0028:00000000, EOI_Proc=Default_Shared_EOI
        Mask_Change_Proc=0028:00000000, IRET_Proc=Default_Shared_IRET

IRQ 14: Hw_Int_Proc=Default_INT, Refdata=00000000
        Virt_Int_Proc=0028:00000000, EOI_Proc=Default_EOI
        Mask_Change_Proc=Default_MASK, IRET_Proc=Default_Iret

IRQ 15: Hw_Int_Proc=_Port_ISR, Refdata=C1A21B2C
        Virt_Int_Proc=0028:00000000, EOI_Proc=0028:00000000
        Mask_Change_Proc=0028:00000000, IRET_Proc=0028:00000000

IRQ 15: Hw_Int_Proc=Default_PM_Share_Int, Refdata=00000000
        Virt_Int_Proc=0028:00000000, EOI_Proc=0028:00000000
        Mask_Change_Proc=0028:00000000, IRET_Proc=0028:00000000
```

#.VSHARE

```
VSHARE version 0110.
000B files opened by sharer.  0001 files open now.
Handle   Pathname (instances) [locks]
C1F9C03C NUL
```

#.VTD

```
0007F0F4   VTD_CB_Offset
C00ACA14   VTD_IRQ_Handle
C4920154   VTD_Focus_VM
00000000   VTD_Timer_At_VM_Speed
00004000   VTD_Current_Count
00004000   VTD_Next_Count
```

```
00004000   VTD_Ring0_Desired_Count
00003FDB   VTD_Lowest_Latch_Val
AF63E000   VTD_Lo_Num_Ticks
00000001   VTD_Hi_Num_Ticks
AF63DF84   VTD_Last_Time_Update
AF632000   VTD_Last_Virt_Int_Time
0000000A   VTD_Reprog_Min_Count
054C001F   VTD_Orig_Int_08_Vector
0010DB40   VTD_Tick_Count
0000E000   VTD_Partial_Tick_Count

Status for VM C1D20154
VM count set to default (0)
Status for VM C4920154, Timer Focus
VM count set to default (0)
```

#.VTDAPI

```
FFFFFFFF   Handle Table
C1738928   Timer List
---------
Node: C1EDE604
0000AB2D   start time
0001ABC2   total periods
00000037   period
00000037   resolution
00000054   instance data
C163B850   time out handle
C163A5C4   callback event
C1D20154   vm handle
01EF017C   callback
C1EDE604   Handle
00000001   Count
00000203   Flags
CCCCCCCC   Ring0 Thread
```

WDEB386 command line parameters:

```
Microsoft (R) Windows 4.0  Kernel Debugger  Version 4.0.4 06/29/95 19:16:07
Copyright (C) Microsoft Corp 1990-1994.  All rights reserved.
WDEB386 [/A] [/B] [/C:1|2|3|4] [/D:"commands"] [/E] {/F:filespec} [/L] [/N] [/R:dddd] {/S:filespec} [/T:hhhh]
[/V[P]] [/X] programspec [parameters]
        /A don't auto-load symbol files
        /B stop debugger during initialization
        /C specifies COM port for debugging terminal
        /D executes a debugger command line
        /E stops at real-mode entry
        /F command line response file
        /I makes debugger invisible to int 41
        /H load wdeb386 as a vxd
        /L suppress line numbers from SYM file
        /N new debugger option defaults
        /R specifies the baud rate for debugging terminal
        /S specifies symbol files to be loaded
        /T specifies the TEFTI or STAT! card base in hex
        /V displays all segment load notifications
        /VP displays only windows segment load notifications
        /X load symbols in XMS memory
```

WDEB386 direct commands:

```
BC [<bp list> | *] - clear breakpoint(s)
BD [<bp list> | *] - disable breakpoint(s)
BE [<bp list> | *] - enable breakpoint(s)
BL - list breakpoint(s)
BP[<bp>] [<addr>] [<passcnt>] ["<bp cmds>"] - set a breakpoint
BR[<bp>] E|W|R|1|2|4 [<addr>] [<passcnt>] ["<bp cmds>"] - set a debug register
C <range> <addr> - compare bytes
D [<range>] - dump memory
DA [<range>] - dump asciiz string
DB [<range>] - dump memory in bytes
DW [<range>] - dump memory in words
DD [<range>] - dump memory in dwords
DG[A] [<range>] - dump GDT entries
DI[A] [<range>] - dump IDT entries
DL[A|P|S|H] [<range>] - dump LDT entries
DP[A|D] [<range>] - dump page directory/table entries
DT [<addr>] - dump TSS
DX - dump loadall buffer
E <addr> [<list>] - enter memory
F <range> <list> - fill  memory
G[S|H||T|Z] [=<addr> [<addr> ...]] - go
H <word> <word> - hexadd
I <word> - input from port
J <expr> [<cmds>] - execute <cmds> if <expr> is true (non-zero)
KA <numargs> - set number of stack dump arguments
K[V|S|B] [<SS:BPaddr>] [<CS:IPaddr>] - stack trace
K[V|S|B]T [TDB] - task stack trace
LG [<mapname>] - list groups in active maps
LM - list linked and active maps
LN [<addr>] - list near symbols
LA [<mapname>] - list absolute symbols in active maps
LS [<addr>] - list symbols in group
LSE <re> - list symbols specified by regular expression
M <range> <addr> - move
O <port> <byte> - output to port
P[N|T|Z] [=<addr>] [<word>] - program step
R[T]|[<reg>] [[=] <word>] - register
S <range> <list> - search
T[A|C|N|S|X|Z] [=<addr>] [<word>] [<addr>] - trace
U [<range>] - unassemble
V[1 | 3] - specify rings to intercept a trap vector
V2 - enable/disable NMI trapping in Ring 0
VL[N|P|V|R|F] - list trap vectors intercepted
VS[N|P|V|R|F] <byte> ... - intercept trap vector, ring 3 only
VT[N|P|V|R|F] <byte> ... - intercept trap vector, all rings
VC[N|P|V|R|F] <byte> ... - stop intercepting trap vector
W [<map name>] - make named map active
WA [<mapname> | *] - add a map to the active list
WR [<mapname> | *] - remove a map from the active list
```

```
X - dump bug report info
Y[?] - display/modify debugger options
Z  - zap the previous INT 1 or the current INT 3 with NOP's
ZD - execute the default command
ZL - list the default command
ZS - set the default command
. - external commands, execute ".?" for help
? - print help menu
? <expr> | [h|d|t|o|q|b].<expr> - display expression
    (h = hex, d or t = decimal, o or q = octal, b = binary)
? "printf string", <expr>, <expr>, ... - printf command
<range> = [<addr>] [<word>] | <addr> [L <word>]
<addr> = [& | #][<word>:]<word> | %<dword> | %%<dword>
<list> = <byte> <byte> ... | "string"
<binary ops> = : | * / MOD + . - << >> < > >= <= AND XOR OR && ||
<unary ops> = &seg #sel %lin %%phy ! NOT SEG OFF BY WO DW POI PORT WPORT

Regular expressions <re>:
. any character, [] character class, [a-z], [^a], etc
* match zero or more, # match zero or one, + match one or more

Supported printf format characters are:
   %%                                          %
   %c                                          character
   %[-][+][ ][0][width][.precision][p][n]d     decimal
   %[-][0][width][.precision][p][n]u           unsigned decimal
   %[-][#][0][width][.precision][p][n]x        hex
   %[-][#][0][width][.precision][p][n]X        hex
   %[-][0][width][.precision][p][n]o           octal
   %[-][0][width][.precision][p][n]b           binary
   %[-][width][.precision][a]s                 string
   %[-][width][.precision][a][p][n][L][H][N][Z]A  address
   %[-][width][.precision][a][p][n][L][H][N][Z]S  symbol
   %[-][width][.precision][a][p][n][L][H][N][Z]G  group:symbol
   %[-][width][.precision][a][p][n][L][H][N][Z]M  map:group:symbol
   %[-][width][.precision][a][p][n][L][H][N][Z]g  group
   %[-][width][.precision][a][p][n][L][H][N][Z]m  map
     a - pointer to a AddrS structure
     H - 16 bit offset
     L - 32 bit offset
     N - offset only
     Z - no address
     p - gets the previous symbol, address or offset
     n - gets the next symbol, address or offset

The values used in the R cmd for changing the flags register are:
   FLAG           SET             CLEAR
   OverFlow       OV              NV
   Direction      DN (Decrement)  UP (Increment)
   Interrupt      EI (Enabled)    DI (Disabled)
   Sign           NG (Negative)   PL (Plus)
   Zero           ZR              NZ
   Aux Carry      AC              NA
   Parity         PE (Even)       PO (Odd)
   Carry          CY              NC
   Nested Task    NT              (toggles)

Prompts are:  > real mode
              - virtual 8086 mode
              # protected mode
```

Changing a Register

```
#r
AX=0000F002  BX=C1D20154  CX=00000008  DX=00000000  SI=C163B230  DI=C1D9F08C
IP=C0123A71  SP=C1A11D84  BP=C1A11F70  CR2=00000080  CR3=00508  IOPL=3  F=-- --
CS=0028  SS=0030  DS=0030  ES=0030  FS=0030  GS=0030 -- NV UP EI NG NZ NA PE NC
0028:C0123A71  RETD
#r ip
IP=3A71
:3a75
```

Changing a Flag

```
#rf
--(RF) --(VM) --(NT) NV(OV) UP(DN) EI(DI) NG(PL) NZ(ZR) NA(AC) PE(PO) NC(CY)
:cy
```

```
#rf
--(RF) --(VM) --(NT) NV(OV) UP(DN) EI(DI) NG(PL) NZ(ZR) NA(AC) PE(PO) CY(NC)
```

Q computes and prints the checksum of WDEB386’s code segment

```
#q
Debugger CS Checksum: 2048
```

Interrupt Vector Assignments (from Intel i486 Microprocessor manual)

Interrupts and exceptions alter the normal program flow, in order to handle external events, to report errors or exceptional conditions. The difference between interrupts and exceptions is that interrupts are used to handle asynchronous external events while exceptions handle instruction faults. Although a program can generate a software interrupt via an INT N instruction, the processor treats software interrupts as exceptions. Hardware interrupts occur as the result of an external event and are classified into two types: maskable or non-maskable. Interrupts are serviced after the execution of the current instruction. After the interrupt handler is finished servicing the interrupt, execution proceeds with the instruction immediately after the interrupted instruction. Exceptions are classified as faults, traps, or aborts depending on the way they are reported, and whether or not restart of the instruction causing the exception is supported. Faults are exceptions that are detected and serviced before the execution of the faulting instruction. A fault would occur in a virtual memory system, when the processor references a page or a segment which was not present. The operating system would fetch the page or segment from disk, and then the processor would restart the instruction. Traps are exceptions that are reported immediately after the execution of the instruction which caused the problem. User defined interrupts are examples of traps. Aborts are exceptions which do not permit the precise location of the instruction causing the exception to be determined. Aborts are used to report severe errors, such as a hardware error, or illegal values in system tables. Thus, when an interrupt service routine has been completed, execution proceeds from the instruction immediately following the interrupted instruction. On the other hand, the return address from an exception fault routine will always point at the instruction causing the exception and include any leading instruction prefixes.

Function	Interrupt Number	Instruction which can cause exception	Return address points to faulting instruction	Type
Divide error	0	DIV, IDIV	yes	FAULT
Debug exception	1	any	yes	TRAP
NMI interrupt	2	INT 2 or NMI	no	NMI
One byte interrupt	3	INT	no	TRAP
Interrupt on overflow	4	INTO	no	TRAP
Array bounds check	5	BOUND	yes	FAULT
Invalid OP-code	6	any illegal	yes	FAULT
Device not available	7	ESC, WAIT	yes	FAULT
Double fault	8	any that generate an exception		ABORT
Intel reserved	9			
Invalid TSS	10	JMP, CALL, IRET, INT	yes	FAULT
Segment not present	11	segment register	yes	FAULT
Stack fault	12	stack references	yes	FAULT
General Protection Fault	13	any memory reference	yes	FAULT
Page Fault	14	any memory access or code fetch	yes	FAULT
Intel reserved	15			
Floating point error	16	floating point, WAIT	yes	FAULT
Alignment check interrupt	17	unaligned memory access	yes	FAULT
Intel reserved	18-32			
Two byte interrupt	0-255	INT n	no	TRAP

General tips and suggestions:

The single most useful command is ‘X’, which displays a considerable amount of debug information on the current state of the operating system.

<control> S sends an XOFF to WDEB386 to stop a display scrolling
<control> Q sends an XON to WDEB386 to resume display output
<control> A repeats the last debug command

NOP is 144 (90h) (if you have to hand-patch an instruction)

WDEB386 is case insensitive and will accept periods in place of the underscore when specifying symbols.
For example:
DD LOGICAL_DRIVE_TABLE
is equivalent to
dd logical.drive.table

Conditional breakpoint syntax example:
bp _foo “j (eax == 0) ‘ln;r’;r;g”
bp _foo sets a breakpoint at _foo. “j (eax == 0) “ makes this a conditional breakpoint, with the condition listed within the parentheses. <‘ln;r’;r;g> are the branches to take. If the condition is true, the first branch set (delimited by single quotes) is taken. If the condition is false, the second command set (after the single quotes) is

taken. So, if _foo is reached, the conditional breakpoint forces WDEB386 to pause and examine EAX. If EAX == 0, the first branch is taken: 'ln;r', show Line Number and Registers. If EAX != 0, the second branch is taken: r;g, show Registers and Go.

Tips on examining code:
See if the symbols are loaded with the LM (list maps) command. If the symbols are NOT loaded, either load them with AUTOSYM.EXE or add the symbol to the list of symbols (either in RUNWDEB.WRF or SYSTEM.INI) and restart Windows 95. If the symbols ARE loaded, use LG <mapname> (list grous in active maps) to list the groups in the symbol table. Use LS <address of group from prior LG command> (list symbols in group) to show labels and structures.

```
#lm
VCACHE is active
VFAT is active
VWIN32 is active
KERNEL32 is active
GDI is active
USER is active
KERNEL is active
IFSMGR is active
DOS386 is active
VOLTRACK is active
SCSIPORT is active
HSFLOP is active
IOS is active
```

```
#lg ios
IOS:
%C00379B4 _LGROUP
%C0200898 _LMGROUP
%C03750A4 _IGROUP
%C0278DC0 _PTEXT
%C0325408 _PDATA
@0406:00000000 _DBOGROUP
%C022DCE8 _RARE
```

```
#ls .lgroup
 1. %C00379B4 IOS:_LGROUP:_LGROUP
 2. %C00A7258 VCACHE:_LGROUP:_LGROUP
 3. %C008AA08 VFAT:_LGROUP:_LGROUP
 4. %C0077B50 VWIN32:_LGROUP:_LGROUP
 5. %C007BDD4 IFSMGR:_LGROUP:_LGROUP
 6. %C1E8C594 VOLTRACK:_LGROUP:_LGROUP
 7. %C1F662C8 SCSIPORT:_LGROUP:_LGROUP
 8. %C1F634B4 HSFLOP:_LGROUP:_LGROUP
Select symbol: 1
%C00379B4 IOS_Control
%C00379EE IOS_Get_Version
%C00379FD IOS_System_Exit
..<symbols omitted>
%C003B80F IOS_Int13_Device_Chain
..<symbols omitted>
%C003DE9A DRPStrTbl
```

```
#db ios.int13.device.chain
%C003B80F IOS:_LGROUP:IOS_Int13_Device_Chain
%C003B80F 5B 18 F6 C1 1A 9C 03 C0-00 00 00 00 00 00 00 00 [...............
%C003B81F 00 00 00 00 00 00 00 1F-9C 05 00 00 00 00 00 00 ................
%C003B82F 00 01 90 0C 00 00 00 00-00 2E 49 2A 20 2D 2D 2D ..........I* ---
%C003B83F 2D 2D 2D 2D 2D 2D 2D 20-44 75 6D 70 73 20 49 2F ------- Dumps I/
%C003B84F 4F 20 53 75 62 73 79 73-74 65 6D 20 73 74 72 75 O Subsystem stru
%C003B85F 63 74 75 72 65 73 2E 20-20 54 79 70 65 20 27 2E ctures.  Type '.
%C003B86F 49 3F 27 20 66 6F 72 20-6D 6F 72 65 20 69 6E 66 I?' for more inf
%C003B87F 6F 0D 0A 00 0D 0A 49 4F-50 20 66 6F 6C 6C 6F 77 o.....IOP follow
```

IOS structures

Dump Physical Drive Table

```
#dd physical.drive.table
%C003B720 IOS:_LGROUP:physical_drive_table
%C003B720  C1F62D78 00000000 00000000 00000000
%C003B730  00000000 00000000 00000000 00000000
%C003B740  00000000 00000000 00000000 00000000
%C003B750  00000000 00000000 00000000 00000000
%C003B760  00000000 00000000 00000000 00000000
%C003B770  00000000 00000000 00000000 00000000
%C003B780  C1F61798 C1F62660 00000000 00000000
%C003B790  00000000 00000000 00000000 00000000
```

```
#.idcb c1f62d78
DCB_BDD_Next                00000000      DCB_ptr_cd              C1F62B38
DCB_next_dcb                C1F633D8      DCB_BDD_BD_Major_Ver    00
DCB_next_logical_dcb        C1F633D8      DCB_BDD_BD_Minor_Ver    00
DCB_next_ddb_dcb            00000000      DCB_BDD_Device_SubType  00
DCB_dev_node                C17DC698      DCB_BDD_Int_13h_Number  80
DCB_device_flags            11008103      DCB_device_type         00
DCB_BDD_Name_Ptr            00000000      DCB_bus_type            01
DCB_apparent_sector_cnt[0]  001FE804      DCB_drive_lttr_equiv    00
DCB_apparent_sector_cnt[1]  00001FE8      DCB_unit_number         80
DCB_apparent_blk_size       00000200      DCB_bus_number          00
DCB_apparent_head_cnt       00000021      DCB_max_sg_elements     11
DCB_apparent_cyl_cnt        000003FE      DCB_io_pend_count       00
DCB_apparent_spt            0000003E      DCB_lock_count          00
DCB_BDD_Sync_Cmd_Proc       00000000      DCB_hardware_unit       00
DCB_BDD_Command_Proc        00000000      DCB_scsi_lun            00
DCB_BDD_Hw_Int_Proc         00000000      DCB_scsi_hba            01
DCB_BDP_Cmd_Queue_Ascending 00000000      DCB_max_sens_data_len   12
DCB_BDP_Cmd_Que_Descending  00000000      DCB_srb_ext_size        001A
DCB_BDP_Current_Flags       00000000      DCB_apparnt_blk_shift   09
DCB_BDP_Int13_Param_Ptr     00000000      DCB_current_unit        00
DCB_BDP_Current_Command     00000000      DCB_blocked_iop         00000000
DCB_BDP_Current_Position[0] 00076DC7      DCB_vrp_ptr             00000000
DCB_BDP_Current_Position[1] 0000076D      DCB_vol_unlock_timer    00000000
DCB_fastdisk_bdd            00000000      DCB_access_timer        00
DCB_actual_sector_cnt[0]    001FE804      DCB_Vol_Flags           00
DCB_actual_sector_cnt[1]    00001FE8      DCB_dmd_flags           00048A00
DCB_actual_blk_size         00000000      DCB_q_algo              01
DCB_actual_head_cnt         00000021      DCB_sig                 4342
DCB_actual_cyl_cnt          000003FE      DCB_reserved1           00000000
DCB_actual_spt              0000003E      DCB_reserved2           00000000
DCB_physical_dcb            C1F62D78      DCB_Exclusive_VM        00000000
DCB_expansion_length        0000007C      DCB_max_xfer_len        00FFFFFF
DCB_Partition_Start         00000000      DCB_cAssoc              00
DCB_vendor_id   DEC
DCB_product_id  DSP3107LS
DCB_port_name   ncrc810
DCB call down list follows:
DCB_cd_io_address C1738A00 DCB_cd_next C1F62B14 DCB_cd_lgn 07 DRP_TSD
DCB_cd_io_address C1E88AF4 DCB_cd_next C1F62EBC DCB_cd_lgn 10 DRP_VSD_8
DCB_cd_io_address C1F67276 DCB_cd_next C003B813 DCB_cd_lgn 14 DRP_NT_MPD
DCB_cd_io_address C0039C1A DCB_cd_next 00000000 DCB_cd_lgn 1F DRP_IOS_REG
```

Dump first 26 entries in logical drive table (each non-zero entry is a DCB)

```
#dd logical.drive.table l26t
%C003B6B8 IOS:_LGROUP:logical_drive_table
%C003B6B8  C1F61798 C1F62660 C1F633D8 C1F61A54
%C003B6C8  C1F62C14 00000000 00000000 00000000
%C003B6D8  00000000 00000000 00000000 00000000
%C003B6E8  00000000 00000000 00000000 00000000
%C003B6F8  00000000 00000000 00000000 00000000
%C003B708  00000000 00000000 00000000 00000000
%C003B718  00000000 00000000
```

```
#.idcb c1f62660
DCB_BDD_Next                C1F62E3B      DCB_ptr_cd              C1E7ED7C
DCB_next_dcb                C1F627A4      DCB_BDD_BD_Major_Ver    00
DCB_next_logical_dcb        00000000      DCB_BDD_BD_Minor_Ver    00
DCB_next_ddb_dcb            C1F61798      DCB_BDD_Device_SubType  00
DCB_dev_node                C17D9344      DCB_BDD_Int_13h_Number  01
DCB_device_flags            1018D017      DCB_device_type         0A
DCB_BDD_Name_Ptr            00000000      DCB_bus_type            02
DCB_apparent_sector_cnt[0]  00000960      DCB_drive_lttr_equiv    01
DCB_apparent_sector_cnt[1]  00000009      DCB_unit_number         01
DCB_apparent_blk_size       00000200      DCB_bus_number          00
DCB_apparent_head_cnt       00000002      DCB_max_sg_elements     11
DCB_apparent_cyl_cnt        00000050      DCB_io_pend_count       00
DCB_apparent_spt            0000000F      DCB_lock_count          00
DCB_BDD_Sync_Cmd_Proc       00000000      DCB_hardware_unit       00
DCB_BDD_Command_Proc        00000000      DCB_scsi_lun            00
DCB_BDD_Hw_Int_Proc         00000000      DCB_scsi_hba            00
DCB_BDP_Cmd_Queue_Ascending 00000000      DCB_max_sens_data_len   00
DCB_BDP_Cmd_Que_Descending  00000000      DCB_srb_ext_size        0000
DCB_BDP_Current_Flags       00000000      DCB_apparnt_blk_shift   09
```

```
DCB_BDP_Int13_Param_Ptr     00000000      DCB_current_unit        00
DCB_BDP_Current_Command     00000000      DCB_blocked_iop         00000000
DCB_BDP_Current_Position[0] 00000000      DCB_vrp_ptr             C1F91494
DCB_BDP_Current_Position[1] 00000000      DCB_vol_unlock_timer    00000000
DCB_fastdisk_bdd            00000000      DCB_access_timer        E7
DCB_actual_sector_cnt[0]    00000960      DCB_Vol_Flags           40
DCB_actual_sector_cnt[1]    00000009      DCB_dmd_flags           00640800
DCB_actual_blk_size         00000200      DCB_q_algo              01
DCB_actual_head_cnt         00000002      DCB_sig                 4342
DCB_actual_cyl_cnt          00000050      DCB_reserved1           00000000
DCB_actual_spt              0000000F      DCB_reserved2           00000000
DCB_physical_dcb            C1F62660      DCB_Exclusive_VM        00000000
DCB_expansion_length        00000004      DCB_max_xfer_len        FFFFFFFF
DCB_Partition_Start         00000000      DCB_cAssoc              01
DCB_vendor_id   GENERIC
DCB_product_id  NEC  FLOPPY DISK
DCB_port_name
DCB call down list follows:
DCB_cd_io_address C1E8C5A0 DCB_cd_next C1F6162C DCB_cd_lgn 05 DRP_VOLTRK
DCB_cd_io_address C1738A00 DCB_cd_next C1F619AC DCB_cd_lgn 07 DRP_TSD
DCB_cd_io_address C0037A74 DCB_cd_next C1F61988 DCB_cd_lgn 12 DRP_RESRVD18
DCB_cd_io_address C1F63749 DCB_cd_next C003B813 DCB_cd_lgn 1B DRP_NEC_FLOPPY
DCB_cd_io_address C0039C1A DCB_cd_next 00000000 DCB_cd_lgn 1F DRP_IOS_REG
```

Running WDEB386 as a VXD

WDEB386 can be loaded as a VXD by adding the following lines to the end of the SYSTEM.INI [386enh] section:

```
debugcom=2
debugbaud=19200
device=wdeb386.exe
device=debugcmd.VXD
LoadDebugOnlyObjs=YES
```

The Stress group's PSGRET.EXE will automatically add these lines to SYSTEM.INI during installation of the symbol files.

```
DebugCom = Sets the com port number.
DebugBaud = Sets the com port baud rate.
DebugCmd = Execute debugger commands.  For commands like "y /n", etc.
DebugSym = Loads symbol files.  Muliple symbol file names can be put on a line separated by spaces and multiple
DebugSym keywords are supported.
```

The DebugCom and DebugBaud switches force all debug output to the specified serial port even when the debugger is not running.

Here is an example from SYSTEM.INI:

```
[386enh]
;;at end of 386enh section..
debugcom=2
debugbaud=19200
device=vdbstruc.exe
device=doozer.exe
device=vwinmm.exe
device=wdeb386.exe
device=debugcmd.VXD
LoadDebugOnlyObjs=YES
;; following command to WDEB386 stops in debugger when DOS386 is about to load
DebugCmd=y /b
debugsym=C:\WINDOWS\SYSTEM\ios.sym
debugsym=C:\WINDOWS\SYSTEM\IOSUBSYS\hsflop.sym
debugsym=C:\WINDOWS\SYSTEM\IOSUBSYS\scsiport.sym
debugsym=C:\WINDOWS\SYSTEM\IOSUBSYS\voltrack.sym
debugsym=C:\WINDOWS\SYSTEM\IOSUBSYS\disktsd.sym
debugsym=C:\WINDOWS\SYSTEM\dos386.sym
```

a-Jeffs "Setting up debuggers"

Connectors:

5 Types:

- 25 Pin Males
- 25 Pin Females, (Non IBM machines)
- 25 Pin Females, no pin 22 (IBM machines)
- 9 Pin, Debug
- 9 Pin, FASync

Uses:

- 25 Pin Males go tothe back of Esprit Debuggers
- 25 Pin Females go to the back of machine's serial port and/or Esprit AUX port

25 Pin Females, no pin 22, are used on IBM serial ports only
9 Pin, Debug, are used on 9 Pin serial ports
9 Pin, FASync, are used on machine to machines connections
(When connecting from machine to machine use 1 FASync and 1 Debug connector)

Cables:
Use Wide RJ-45 cables (Contain Grey,Orange,Black,Red,Green,Yellow,Blue, and Brown wires)

Setting up the com port for data transmission:
Type: **mode com1:19.2,N,8,1** OR **mode com2:19.2,N,8,1**

The MSDOS MODE command supports 19,200 baud on very few PCs besides the IBM PS/2 family. To reliably and easily set the baud rate of all to 19200,n,8,1 (the default settings for WDEB386), use COM192.COM (found on \\pyrex\user!percyt). COM192 does the following: enumerates all serial ports it can find on PC, and lists their port number (COM1 to COM4) and base address; sets each port to 19200,n,8,1; outputs a short string from each, which should be displayed on any debug terminal or terminal emulation application. COM192 shows the letter 't' to the left of each port it was unable to write a string to.

Testing data transmission:
Type: dir > com1 OR dir > com2 OR run COM192 COM1
(or COM2, COM3, COM4) will poll the specified serial port and will echo any text entered in at the other terminal until either the string 'CONNECT' is received or a key is pressed on the COM192 end.

Esprit Terminals:
Press Shift and Setup Assist Key to enter into setup program. To make the debugger split screen, edit configuration menu and change to Dual Host/Dual session and Dual video mode. Also, make sure that the data transmission is set to 19.2.

Machine to Machine:
When setting up from machine to machine, one machine must be running debug software (i.e. Kermit, RTerm, or Terminal). Make sure connectors are 1 debug and 1 FASync.

Note: Sometimes transmission is stuck and Esprit terminal must be turned off and on again.

Aarono (October 28, 1992) Using CodeView for Windows under WDEB386

Start Winodws95 under WDEB386 with the /b option:
RUNWDEB.BAT

```
@wdeb386 %1 /n /x /c:2 /r:19200 /f:runwdeb.wrf ..\win.com /b dos386.exe
```

WDEB386 loaded as a VxD
DebugCmd=y /b

When the breakpoint happens, from your debug terminal type:

{disables ring 3 trapping of int1}

```
#v1
Rings trapped for int 1 - 0 1 2 3 V
? -3 -
```

{disables ring 3 trapping of int3}

```
#v3
Rings trapped for int 3 - 0 1 2 3 V
? -3
```

This technique to disable int 3 is also usable for debugging apps that scatter int 3's through their code, such as Xtree Gold:
XTREE hooks Int 3 and uses it as a way to talk to himself. You must disable int 3 before running this app!

Raymondc (July 28, 1993) How to debug a Crash_Cur_VM

Crash_Cur_VM is typically reached when a VxD GP faults. To find out who is *really* at fault (sorry), type "dd ebp+24 l2". You should get something like

```
        0030:xxxxxxxx 8yyyyyyy xxxx0028
                      ^^^^^^^^     ^^^^
                       EIP          CS
```

If CS is 0028, you should type

```
        ln %8yyyyyyyy
```

to find out who **really** crashed.

DavidDs (January 1994) debugging tips handout

I've faulted and can't get up

The most important things to do are 'ln' and get the address, and get the last debug messages spit out. Note the text on the fault box. This is important to differentiate between a Page and GP fault. Hit the <Cancel> button. Report where the code is via the 'r' command. 'ln' should give a symbol and line number. Look at the prompt and see where you are. A '-' prompt means v86 mode. A '#' means pmode. Use 'k' for a stack trace to get the bug owner. If selector 28, you are in a VxD. If in v86 mode or VxD, do '.r'. The address tells you who got you to this location. Do a 'g' <address>, get a stack trace ('k') from here and you can get the owner of the bug.

I'm really hung and can't get it going.

Use <Control><C>. Did it work? If not, probably dead in VxD land. Try <Control><Sys Rq> ("Print Screen" key on hung PC); '.r' to see where you are; 'g' <address> to get there; 'k' stack trace to see bug owner. If you are in V86 mode, you may want to try '.r' again to see what pmode component you get to.

Kensy (January 07, 1994) VDBSTRUC - Generic structure-dumping VxD

We (RayEn & I) have been using VDBSTRUC in the GDI group for a long time & it's proved to be useful (and stable) so I've checked it in to tools\binw and toolsrc\vdbstruc. It's important to note this utility is not in any way specific to GDI - it works on any structure that can be obtained from an ASM listing file.

Some notes:
- The makefile in toolsrc\vdbstruc doesn't work as is. The clue is it did work when the directory lived in dos\dos386\vxd\vdbstruc. But if you know how to, say, record every episode of the Simpsons you should be able to fix up the makefile.
- The mkstruc utility only works on ASM listing files. The .str file format is super-simple so if you, say, drive a Q45a and happen to know how to parse browser info, you could write a better utility to generate struc files.
- Take a look at subgdi.mk, struc.asm to see how it's integrated in the GDI project.
===

Most of you have seen (and maybe used) VDBSTRUC, a wdeb extension that dumps structures. I've taken the liberty of integrating this into the GDI project.

INSTALLATION
==============
1) ssync to depend.mk,incs.asm,subgdi.mk in GDI directory
2) ssync to mkstruc.exe in dev\tools\binr
2) copy vdbstruc.* from \\pyrex\user\kensy into your system directory
3) add the following lines to the [386enh] section of system.ini:

```
        device=vdbstruc.386
        StrucFile=gdi.str
```

MAKING THE STRUC FILE
======================
1) cd DEBUG (or cd RETAIL)
2) nmake struc

This will build gdi.str which you then copy to your system directory.

ADDING NEW STRUCTURES TO GDI.STR
====================================
'nmake struc' assembles incs.asm which includes most of the inc files.
If your include file is not included by incs.asm follow the applicable directions:

ASM include files:
1) add the include line to incs.asm
2) check out depend.mk
3) cd DEBUG & type nmake gdepend

C include files (example uses path.h):
1) copy path.h path.inc
2) check out depend.mk; cd DEBUG & type nmake gdepend
3) delete path.inc in GDI directory
[Steps 1 & 3 are needed because includes.exe skips includes it can't open]
Note: h2inc will convert the .h file. It is very fussy. For instance it requires the { to be on the same line as the typedef. For this reason sort.h & spfile.h weren't added to incs.asm.
4) Type nmake struc to get the updated struc file.

USING VDBSTRUC
================
Here's a brief summary: I will send out more detailed mail when this is integrated into DEBUGCMD.386.
You can list the available structures by typing .KL. The current structure is marked with an asterick.
There is a concept of a default structure and a range of fields. When an address is dumped it is displayed using the default structure & fields. To specify the current structure parameters:

```
.KS strucname.field1-field2
```

You can specify just a structure name, a structure and a single field, or a structure and a range of fields. Each of these entries could also be a prefix followed by '*' and the debugger will list those structures/fields starting with the specified prefix.

Typing just .KS displays the current default settings.
To dump a block of memory:

```
.KD addr
```

where addr is any valid wdeb386 expression. One caveat: ALWAYS specify a segment & offset (unlike the d* commands which default to the previously dumped segment)
To dump memory using a non-default structure:

```
.KD [strucname.field1-field2]addr
```

strucname.field1-field2 follows all the same rules mentioned in .KS.

Brian Smith (February 7, 1994) new page fault logging debugger commands

By default, the last 100 page faults are getting logged into a memory buffer. They can be dumped with the .MF debugger command. The most recent faults are first. The output looks like:

```
3f007b2a tid=0007 cs:eip=0137:3f007b2a mpr:.text:?PlaceDialog@@YGXPAXH@Z
401da000 tid=0007 cs:eip=028f:00000ad7 dibeng:_TEXT:BB_CopyBlock
401d93c0 tid=0007 cs:eip=028f:000003e3 dibeng:_TEXT:RealizeBitmap
```

The first number is the linear address faulted on. "tid" is the thread id of thread that faulted, and cs:eip is the code that caused the fault.

".MFF" will cause the faults to be logged to the debug terminal as they happen in the same format, as well as logged to memory. ".MFB" will cause each fault to stop in the debugger on the faulting instruction(when using .MFB, ignore the message "Type "GT" to terminate...", it is a by-product of the way I stop in the debugger. "GT" will cause you to process the fault and continue, not terminate). .MFF and .MFB a second time will turn off each feature.

By default, all page faults are logged, but you can also ask to just log a particular address range. The syntax is ".MF <starting linear address> L<number of bytes in range>". So, if you want to see if USER's _RARE segment really contains rare code, you can find out its base linear address and size (by using the DL command: "06ef Code Bas=40130360 Lim=00002d7f DPL=3 P RE A UV")and type ".MFB 40130360 L2d7f". That will cause you stop in the debugger each time you get a fault on that segment.

A useful related command is ".MN". It will cause all pageable memory to be marked not present, so when you hit "G" you will get a page fault on every page touched.

Raymondc (March 26, 1994) DOS call tracing for DOS apps

Ron Radko and I have been needing one of these puppies for weeks, and I needed one again today, so I finally broke down and just wrote one. This works only for DOS apps in V86-mode. If you have to debug DOS calls from a DPMI app, all I can do is extend my sympathies.

To install, copy \\pyrex\user\dtrace.exe into your system directory and either add `device=dtrace.exe' to your system.ini or just run it (Dtrace.exe is now a dynaload VxD). Then `.dtrace' will call up the DOS tracing menu.

```
        [1] V86 Int 21h tracing is OFF
        [2] Weenie call tracing is OFF
        [3] Halting on errors is OFF
        Toggle what? (ESC to exit)
```

If you turn on Weenie call tracing, then you will get traces for calls like "Query keyboard character ready" or "Display one character to stdout". If you turn on Halting on errors, you will also halt if a read request reaches EOF or a write request returns disk full. DOS does not consider these to be errors, but apps usually go nonlinear when faced with one of these conditions, so I stop on them anyways. When Halting on errors is enabled, you hit an `int 3' when a DOS call returns error. For convenience, `z'apping the `int 3' is equivalent to navigating through the menus and disabling "Halting on errors".

Briansm (April 28, 1994) running out of handles

The default set of symbols will fit fine on a 8meg machine. The problem is running out of himem.sys's XMS handles. This can be solved by removing symbols from runwdeb.wrf or (better yet) by adding a "/numhandles" switch to your himem.sys, like so:

```
DEVICE=C:\CHICAGO\HIMEM.SYS  /numhandles=60
```

If we don't have enough XMS handles, the vxd loader isn't able to allocate all of the available RAM on the machine. The /L switch to wdeb386 will not help this problem, though it will sometimes hide it by allowing your system to boot even if the vxd loader hasn't been able to allocate all of your machine's RAM.

DavidFl(April 29, 1994) Tips for debugging for registry problems

Actual registry corruption is rare. I absolutely want to know about these cases. You will know if you get registry corruption because:
1) You are told when you boot to DOS
2) IFSHLP.SYS and/or HIMEM.SYS are not found when booting to DOS.
3) When running wdeb386 you will encounter an int 3 in the real mode registry code that loads before you hit the break when using the /b option on wdeb. The routines are _rlFHValid, _rlKHValid and _rlDBValid.
4) Your machine boots but just after the logon dialog you get another dialog that reports a corrupt registry.

Invalid data in the registry is also fairly uncommon - but usually fatal. It is hard to determine exactly if a bug is linked to the registry or not. Missing data from the registry is very common and most components can handle this.

Here is what I would do to try to narrow down the problem.
1) Run Rterm with the debug build of Chicago.
2) runwdeb with the /b option ot break in protect mode.

3) type ".VMM s", this will try on VMM verbose information ie. VxD messages. There are a lot of messages but they will help narrow down the crashing VxD or component. The output will look like:

```
Dev call Sys_Critical_Init to CONFIGMG OK
Dev call Sys_Critical_Init to VCDFSD   OK
Dev call Sys_Critical_Init to IOS      IOS_Sys_Critical_Init + f7 is calling the memory manager incorrectly OK
Dev call Sys_Critical_Init to PAGEFILE OK
Dev call Sys_Critical_Init to PAGESWAP OK
```

Note that the messages that you normally see will be displayed between the "Dev call <message> to <VxD>" string and the "OK" that follows (ie, IOS problem above).
** This may help determine what component is crashing on startup and what message it's handling **

4) Next, to determine if the registry is involved, reboot and either set a break point just before the crash or "control-C" into the debugger just before the crash. At this point you want set the debugger up to watch for error return conditions from the registry code that might be related. To do this, enter the following information into the debugger:
a) type "ls kregclosekey*" and the address of both the real mode and protect mode _KRegCloseKey will be displayed. Looks like:

```
1##ls kregclosekey*
@1486:00005472 _KRegCloseKey
%c0259a27 _KRegCloseKey
1##
```

b) You want to set a break point on the protect mode _KRegCloseKey-5. So set it like (just subtract 5 from the address):

```
1##bp c0259a22 "j eax==0 g;? '  Failure _KRegOpenKey'#eax;dd ebp+8 l 1;g"
```

This command will instruct the debugger to only break when an ID string in the registry is not found. It will then print out the error code (eax) and the HKEY (ebp+8):

```
Failure _KRegOpenKey 0000:000003f3
0030:c3a87e94  80000005
```

note that 3f3h means ERROR_CANTOPEN and 800000005 means HKEY_CURRENT_CONFIG (See vmmreg.h in ..\dev\inc).

c) The ID string can be printed with the following command at
_KRegCloseKey-4:

```
1##bp c0259a23 "J EAX==0 G;da dw(ebp+c);dd dw(ebp+10) l 1"
```

Output will look like:

```
0030:c0337bac security\provider
0030:c0337bcc  00000000
```

Note that the first line indicates the ID string and the second line is the value written back to the calling app.

Now, if a value is messed up....
d) To get information about the values that are not found type:

```
1##bp _KregCreateDynKey-5 "J EAX==0 G;? '  Failure _KRegQueryValue'#EAX;DD
EBP+8
 L 1;G"
```

and

```
1##bp _KregCreateDynKey-4 "J EAX==0 G;DA DW(EBP+10);? '  End Value';G"
```

The result will be:

```
  Failure _KRegQueryValue 0000:000003f4
0030:c0014544  c2657740
0030:c00145ec FRIENDLYNAME
  End Value
```

Where the first line indicates the error code, the second line shows the key (ie HKEY_* or dynakeys) the third line shows the KeyValue that the component was looking for. The fourth line is just a marker for symmetry.

e) Save the information to a file and get it to the appropriate person. That person will most likely be the developer of the offending VxD or component that crashed.

NOTE: If you enter these break point commands on startup it will slow down you're boot process. There are a lot of components that receive failure return codes. Also, using ".vmm s" will also slow down you're boot. Set the break points close the the offending module.

tips:
- *NEVER* delete the offending *.dat file. Always rename it.
- Don't add new hardware or change the machine's configuration until the problem is understood.

Jeffpar (May 03, 1994) debugging without a terminal

If you have to debug something in chicago and there's no debug terminal or rterm available (or is, but it isn't working), try putting DebugVGA=1 under [386Enh] in system.ini. This will allow you to debug on the same monitor. Use F12 instead of Ctrl-C to break in. Use pgdn and pgup to scroll the screen. Use F4 to view the original screen contents. It's not guaranteed to work with all video adapters and display modes, but chances are good if you just installed (since that means you're still using the standard 16-color vga display driver). WD chipsets work well, especially those in Toshiba portables. DebugVGA support is in the M6 wdeb386.exe, but the latest version (with the F4 key and better WD support) is on \\pyrex\user\jeffpar. Currently, I don't recommend any windows display driver other than vga.drv; I've only added support for standard VGA stuff. It works on the three systems I've tested so far (Compaq VGA, Western Digitial, and S3), all running vga.drv. If it works for you, be happy. If it doesn't, move on. Or, spend some of your spare time adding support for your particular h/w and display mode(s); the screen switching code is in debug.asm, in the vxd\debug directory. Feel free to send me email telling me it does or doesn't work, but don't expect me to do anything about it. :)

The COM port is still used up until device-init, when the debug vxd switches to a blue debugger screen. Press enter and the display will switch back to your original screen. In this mode, the debug vxd registers F12 (that's plain ol' F12, not Ctrl+Alt+F12) as the preferred debugger hotkey. I/O reverts back to the COM port during sys-critical-exit. The screen should switch automatically whenever the debugger requests input (like after traps, RIPs, etc). "G" commands restore the screen. "P" and "T" commands leave you in the debugger's context. I wouldn't recommend stepping over calls that might write to the screen. Eventually I'll add switching forms of the trace commands, as well as F4 to view the windows context, and other widgets like command-line history. One that's already more or less done is screen scroll-back; use PgDn and PgUp to view stuff that's scrolled off the debugger's screen.

There's a copy of the new wdeb386.exe in \\pyrex\user\jeffpar. The changes have been checked in for build 86. Thx to mikem for misc. advice and code reviewing.

Raymondc (October 15, 1994) Interpreting Blue Screens

This mechanism is not restricted to VxDs. It also works for Win16 modules. Resolving Win32 crash dialogs is slightly different, but more obvious. (Left as an exercise.)

1. Locate the symbol file corresponding to the VxD, build, and flavor. In this example, it is VxD=VCOMM, build=205, flavor=retail.
2. Type "viewsym vcomm.sym" to view the symbols for that VxD.
3. Look up object 1. (The number in parentheses is the object number.)
4. Look for the symbol nearest offset 1B9. (The "+ xxxx" is the offset inside the object.)
5. You find VCOMMW32_Completion_Routine at offset 1A8.
6. Do some subtraction.
7. The answer is that you crashed at VCOMMW32_Completion_Routine + 11.
8. Include this information in your bug report. (Crucial!)
9. (Optional) Look up the code and find out what is wrong.

In this case, you find that the faulting instruction is the one that fills in the OVERLAPPED structure. The reason is that the comm port was closed while there was still outstanding overlapped I/O on it. Stop doing that.

10. (Optional) Look at the SLM logs to see what happened here recently.

``Fixed" in build 206 (more accurately, ``hacked") by ignoring I/O completion on ports that have been closed. You will never find out whether your overlapped I/O completed successfully. Tough.

Note! Now that everybody knows what to do, I do not ever want to see a blue-screen bug report again that doesn't at least go through step 8.

Kurte (October 26, 1994) Debug school 101... (Don't trust symbols from explorer.exe)

Today I have received yet another bug assigned to the shell, when the user did a K and found some symbol in explorer.exe. This symbol was totally bogus as the GP fault was in winhelp.exe.

If you get a GP fault or page fault, or rip.... And you do an LN or a K and it shows you a symbol from an EXE these symbols are only valid if the current process is that exe. A simple way to find this out is to do a .W
command which will show you all of the window processes. The one with the * next to is the current process. Another approach is to do a "wr explorer" and then do the LN or k command and in these cases it will show the name of the exe that the fault is in...

Mikem (December 1, 1994) Loop through a linked list in WDEB386

```
zs "?'%08.8x', eax; reax = dw (%(eax+14));zd"
```

will allow you to loop though a linked list (in this example the next pointer is at offset 14) printing each packet on the list. It does trash EAX, but that can be restored. You can also replace the "zd" with a "j eax != 0 'zd' " and the it will stop at the end or some other condition to stop on. You need to set EAX to the address of the first record and execute "zd".

Pierreys (December 10, 1994) How to debug a CONFIGMG problem

1) If you do ".y<enter>v", I will give you a status which on the last line will tell you whether I think this is a configmg problem or not. Basically, I check if we are inside one of CONFIGMG's function. This will not necessarily means it is a CONFIGMG, as we could be in a driver or vmm/shell/vxdldr or even ring3 code.
2) If you do ".y<enter>k", you will get the trace of configmg last calls that made it to wherever your are when you faulted. Here is a sample:

```
00000000234F5E77> return;
00000000234F5E6D> return;
00000000234F5E56> DeallocateAppyTime(C163A6BC);
00000000234F5E44> return;
00000000234F5ACE> return(00000001);
00000000234F5ABF> return(00000000);
```

Note1

```
000000002349A3F0> return(00000001);
000000002349A3BC> ReceiveMessage(00000219,00008001,80148B80,00000000);
0000000023499735> CMBroadcastMessage(00008001,C1597FC4,00000001);
0000000023499723> SendDeviceMessage(HTREE\RESERVED\0,00008001,00000001);
000000023498E3A> ProcessAppyTime(C163A6BC);
000000023498E23> AsyncCallBackAppyTime(C163A6BC);
```

Note that this sample trace is what you will typically get once you are up and running: It shows CONFIGMG last thing was a query remove of the **** devnode which it does every so often. If the trace start with a >return like the sample, then it is not a configmg problem (in fact that is how ".y<enter>v" works. If the trace is where Note1 is, then we died in ring3 processing the query removal I sent. The next step is to do ".vmm c" and ".hbsm" to find who is the ring3 culprit.

If the trace has a
```
"CallHandlerOfDevNode(value,value,devnode_name1,devnode_name2,value)"
```
which was not returned to, then devnode_name1 is the culprit.

Kkennedy (January 7, 1995) Turn off RIPs in debug

On the debug terminal at the # prompt type:
```
e DebugOptions 0 0c
```

A-Jimho (February 10, 1995) Tip: using Hypertrm in terminal mode

Lotsa folks haul out terminal.exe rather than use HT which does have a "hidden" terminal mode. Programs > accessories> HT > create a connectoid to a BBS or phone number of your choice >open connectoid (or go to properties if connectoid is open) > notice item: "connect using" usually contains modem name > open drop down list > notice "direct to com1, direct to com2, etc" > select appropriate com port > you are in terminal mode > issue AT style commands or init string to modem (example: dial out with ATDT nnnnnn) > reset to previous modem name when done with terminal mode.

Raymondc (February 19, 1995) New Tool - LN.EXE lets you debug without symbols

\\pyrex\user\raymondc\ln.exe lets you debug without symbols by implementing the debugger's "ln" command off-line. It is meant to be run in conjunction with rterm. This allows us to debug problems remotely without having to ship the site any symbol files. Suppose you need to debug a problem on build 330 retail. Start a DOS box. Make the current directory equal to the directory where all the symbols are. For example,
```
                net use z: \\hitme\weekly
                z:
                cd \build330\psgret
                ln
```

Now suppose the debugger gives the address
```
        C00CF314:VMPOLL(01) + 12f
```

Highlight this entire phrase in rterm, then select Edit -> Copy to copy it to the clipboard. Then go to the DOS box that is running ln and click Paste. Then press Enter. ln will parse the input, load vmpoll.sym, and look up the symbol in object 1 offset 12f. Eventually, it will print
```
                File: ..\vmpoll.asm - Line: 1712
                VMPoll_System_Idle + 2e
```

Right now, ln supports only VxD symbol files, but Win32 and Win16 support are in the works, as well as a less klunky interface. At which point nobody will have an excuse of "But you didn't load foobar symbols" for not debugging a problem.

\\pyrex\user\raymondc\ln.exe has the following new features.
- Can provide path for symbol files on command line.
```
        ln \\hitme\weekly\build408\psgret
```
- You can provide multiple directories by separating them with semicolons. You can say "lnbuild 408" with the following batch file:
```
        ln \\hitme\weekly\build%1\psgret
```
- Can adjust the path at runtime via the `.d' command.
- Now with Win16 support. Type USER!(01) 07bf:2341 and get
```
        File: ..\wmflash.c - Line: 44
        IFLASHWINDOW + 9b
```
- Can resolve symbols back to object numbers and offsets.
```
        Type this             To get this
        createwindow          USER(0B):068b
        wep                   (all WEP functions in the cache)
        netdi:wep             NETDI(1B):0156
        _pagefree             VMM(12):0170
        nwlink:_rip_socket    NWLINK(01):beb8
```
On the to-do list is resolving the object number and offset all the way back to an address.
- Symbol files are added to the cache when they are first touched. For example, the first time you mention netdi, netdi.sym will be loaded into the cache.

You can load a file into the cache by typing `.l file' and you can unload a file with `.u file'. To view the cache, type `.w'. Loading symbol files into the cache manually is important for user and gdi, because user.sym is the stripped version of the USER symbol file. Type `.l userf' to load the full USER symbol file. Loading symbol files into the cache manually is also important for kernel, because there is no kernel.sym; the file is krnl386.sym. So type `.l krnl386' before trying to resolve kernel symbols.

Raymondc (February 26, 1995) New tool - flog - For people who run without a debugger

\\pyrex\user\raymondc\flog.exe is a sort of "Dr Watson for ring 0" program. (It was originally named `kato', but `flog' sounds better.) When a VxD GP faults, flog wakes up and records a FAULTLOG.TXT file in your Windows directory. To install flog, add the line
```
        device=flog.exe
```
to the [386Enh] section of your system.ini. Flog's memory footprint is 112 bytes of locked code and data; the rest is pulled in only when a fault actually happens. I strongly advise installing flog if you do not with with the debugger installed. That way, when you hit a blue-screen fatal error, you will have something to show me

instead of just filing a bug saying, "I got this message" and which I will just resolve as "Not repro due to insufficient infomation provided". Here's an annotated fault log. The details of the log will, of course, vary. If there is something you would like to see recorded in the log, send me email and I'll consider adding it.

```
Fault log generated on 1995.02.25 14:04:17   [date/time]

These gentlemen must here be reminded of their error.
               -- The Federalist Papers      [inspirational quote]

gs   00000030                                [register contents]
fs   00000030
es   00000030
ds   00000030
edi  c1709abc -> 01 14 51 00 38 9d 70 c1     [if the register looks
esi  c162e4c8 -> 10 f7 62 c1 dc e1 62 c1      like a pointer, some
ebp  c153df70 -> 27 1c e4 87 47 01 00 00      bytes are dumped, too]
esp  c153daec -> 08 50 40 c0 08 50 40 c0
ebx  c1d20154 -> 62 18 00 00 00 00 c0 c1
edx  ffffffb0
ecx  00000002
eax  00000000
flt  0000000e                                [Fault 0E = page fault]
eip  c0005384 = VMM(01) + 00004384 -> 8b 7a 10 8b 32 2b 7e 08
cs   00000028   [^^ if the address looks like a VxD, then symbolic
                   information is extracted in LN.EXE format]

                                             [stack dump]
c153daec c0004528 = VMM(01) + 00003528 -> 8b 3d 40 f8 01 c0 8f 47
c153daf0 00000000
c153daf4 c162e858 -> e6 52 00 c0 24 81 01 c0
c153daf8 00000000
c153dafc c0001400 = VMM(01) + 00000400 -> 80 3d 85 a2 01 c0 00 74
c153db00 00000030
c153db04 00000030
c153db08 c1709abc -> 01 14 51 00 38 9d 70 c1
c153db0c 00000000
c153db10 c153df70 -> 27 1c e4 87 47 01 00 00
c153db14 c153db28 -> 70 df 53 c1 40 db 53 c1
c153db18 c1d20154 -> 62 18 00 00 00 00 c0 c1
c153db1c 00000000
[... and so on ...]

End of fault log
```

Raymondc (June 23, 1995) Yet another VxD troubleshooting tool

Copy \\pyrex\user\raymondc\convmem.vxd into your System directory and add the line

```
device=convmem.vxd
```

to the [386Enh] section of your system.ini. After booting, you will have a file CONVMEM.TXT in your Windows directory, which gives a byte-for-byte accounting of all the conventional memory that VMM32 allocated. (The stuff that shows up in a "mem /d" as belonging to VMM32.) This will help PSS figure out why some users have so little available conventional memory in a DOS box.

Doom

vcp1 to disable trap1 in Doom