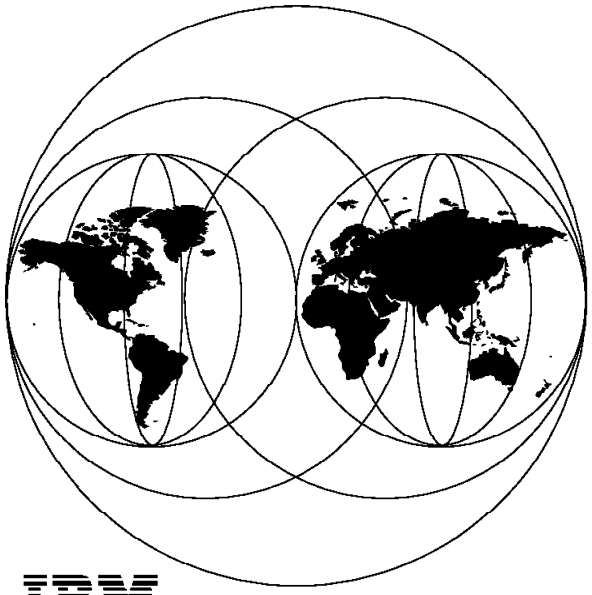
International Technical Support Organization

SG24-4641-00

OS/2 Debugging Handbook - Volume II
Using the Debug Kernel and Dump Formatter

February 1996



IBM

International Technical Support Organization
Boca Raton Center

International Technical Support Organization

SG24-4641-00

OS/2 Debugging Handbook - Volume II
Using the Debug Kernel and Dump Formatter

February 1996

Take Note!

Before using this information and the product it supports, be sure to read the general information under “Special Notices” on page xv.

First Edition (February 1996)

This edition applies to IBM OS/2 Warp Version 3.0.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

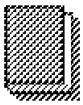
An ITSO Technical Bulletin Evaluation Form for reader's feedback appears facing Chapter 1. If the form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. JLPC Building 014-1 Internal Zip 5220
1000 NW 51st Street
Boca Raton, Florida 33431-1328

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

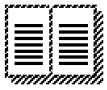
The OS/2 Debugging Handbook Library

The following information describes the four volumes that comprise the OS/2 Debugging Handbook library. The graphic of the opened book denotes the volume that you are currently reading.



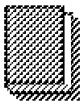
Volume I, *Basic Skills and Diagnostic Techniques, SG24-4640.*

This volume introduces the concepts of debugging with practical examples. Also contained in this book is a CDROM version of the entire library, which is viewable via the OS/2 INF View utility.



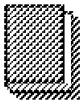
Volume II, *Using the Debug Kernel and Dump Formatter, SG24-4641.*

This volume provides necessary information to set up and use the Kernel Debug and Dump Formatter tools. Also this guide serves as a command reference for these products.



Volume III, *System Trace Reference, SG24-4642.*

This volume includes all system tracepoints contained within OS/2.



Volume IV, *System Diagnostic Reference, SG24-4643.*

This volume provides details of internal structures used by OS/2.

Abstract

This publication is volume two, which is one of four volumes that together provide information and reference materials intended to help perform OS/2 debugging.

This volume provides details on how to set up and use the OS/2 Kernel Debug and Dump Formatter utilities. A Kernel Debug and Dump Formatter command reference is a major part of this book.

This document is intended for use by service personnel, system programmers and software developers.

(299 pages)

Contents

Figures

Tables

Special Notices

This publication is intended to help service personnel, system programmers and software developers to understand the concepts and application of debugging techniques. The information in this publication is intended as a supplement to already published specifications of any programming interfaces that are provided by IBM Warp OS/2 Version 3. See the PUBLICATIONS section of the IBM Programming Announcement for IBM Warp OS/2 Version 3 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (VENDOR) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

IBM	OS/2
Presentation Manager	Workplace Shell

The following terms are trademarks of other companies:

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Windows is a trademark of Microsoft Corporation.

MicroFocus Cobol	MicroFocus Corporation

Other trademarks are trademarks of their respective companies.

Preface

This volume of the OS/2 Debugging Handbook Library details the setting up of the OS/2 Kernel Debug and Dump Formatter utilities. Details of commands used by both utilities are presented and explained in this book. Aided with the other volumes in this series, the trained user will be able to perform debug operations on OS/2 systems.

This document is intended for use by service personnel, system programmers and software developers.

How This Document is Organized

The document is organized as follows:

- Chapter 1, “Kernel Debugger User Guide”

  This chapter describes the set up and use of the Kernel Debugger.

- Chapter 2, “Dump Formatter User Guide”

  This section provides information needed to use the Dump Formatter.

- Chapter 3, “Kernel Debugger and Dump Formatter Command Reference”

  This section details internal and external commands used with the Kernel Debugger and Dump Formatter.

Related Publications

Throughout this book we assume the availability and familiarity with two co-requisite publications:

- *The INTEL486 Microprocessor Programmer's Reference Manual*, ISBN 1-55512-159-4
- *The Intel Pentium Family User's Manual, Volume 3: Architecture and Programming Manual*, ISBN 1-55512-227-2
- *The Design of OS/2 by H.M. Deitel and M.S. Kogan*, ISBN 0-201-54889-5

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this document.

- *The OS/2 Technical Library Control Program Programming Reference Version 2.00*, S10G-6263-00
- *OS/2 2.0 Proc Lang 2/REXX Ref*, S10G-6268-00
- *OS/2 2.0 Proc Lang 2/REXX User Guide*, S10G-6269-00
- *OS/2 WARP Control Program Programming Guide*, G25H-7101-00
- *OS/2 WARP Control Program Programming Ref*, G25H-7102-00
- *OS/2 WARP PM Basic Programming Guide*, G25H-7103-00
- *OS/2 WARP PM Advanced Programming Guide*, G25H-7104-00
- *OS/2 WARP GPI Programming Guide*, G25H-7106-00
- *OS/2 WARP GPI Programming Ref*, G25H-7107-00

- *OS/2 WARP Workplace Shell Programming Guide*, G25H-7108-00
- *OS/2 WARP Workplace Shell Programming Ref*, G25H-7109-00
- *OS/2 WARP IPF Programming Guide*, G25H-7110-00
- *OS/2 WARP Tools Reference*, G25H-7111-00
- *OS/2 WARP Multimedia App Programming Guide*, G25H-7112-00
- *OS/2 WARP Multimedia Subsystem Programming*, G25H-7113-00
- *OS/2 WARP Multimedia Programming Ref*, G25H-7114-00
- *OS/2 WARP PM Programming Ref Vol I*, G25H-7190-00
- *OS/2 WARP PM Programming Ref Vol II*, G25H-7191-00
- *Technical Reference - Personal Computer AT*, Part Number 1502494
- *PS/2 and PC BIOS Interface Technical Reference*, Part Number 68X2341

International Technical Support Organization Publications

- *OS/2 Warp Connect*, GG24-4505
- *OS/2 Warp Generation, Vol.1*, SG24-4552
- *OS/2 Warp Version 3 and BonusPak*, GG24-4426
- *Multimedia in Warp*, GG24-2516
- *The Technical Compendium Volume 1 - Control Program*, GG24-3730
- *The Technical Compendium Volume 2 - DOS and Windows Environment*, GG24-3731
- *The Technical Compendium Volume 3 - Presentation Manager and Workplace Shell*, GG24-3732
- *The Technical Compendium Volume 4 - Application Development*, GG24-3774

A complete list of International Technical Support Organization publications, known as redbooks, with a brief description of each, may be found in:

International Technical Support Organization Bibliography of Redbooks, GG24-3070.

To get a catalog of ITSO redbooks, VNET users may type:

```
TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
```

A listing of all redbooks, sorted by category, may also be found on MKTTOOLS as ITSOCAT TXT. This package is updated monthly.

How to Order ITSO Redbooks

IBM employees in the USA may order ITSO books and CD-ROMs using PUBORDER. Customers in the USA may order by calling 1-800-879-2755 or by faxing 1-800-445-9269. Most major credit cards are accepted. Outside the USA, customers should contact their local IBM office. Guidance may be obtained by sending a PROFS note to BOOKSHOP at DKIBMVM1 or E-mail to bookshop@dk.ibm.com.

Customers may order hardcopy ITSO books individually or in customized sets, called GBOFs, which relate to specific functions of interest. IBM employees and customers may also order ITSO books in online format on CD-ROM collections, which contain redbooks on a variety of products.

ITSO Redbooks on the World Wide Web (WWW)

Internet users may find information about redbooks on the ITSO World Wide Web home page. To access the ITSO Web pages, point your Web browser (such as WebExplorer from the OS/2 3.0 Warp BonusPak) to the following URL:

```
http://www.redbooks.ibm.com/redbooks
```

IBM employees may access LIST3820s of redbooks as well. Point your web browser to the IBM Redbooks home page:

```
http://w3.itsc.pok.ibm.com/redbooks/redbooks.html
```

Acknowledgments

The authors of this book are:

Pete Guy
IBM SDO, Austin

Richard Moore
IBM PSP EMEA

Redbook project developed by:

Tim Sennitt
ITSO Boca Raton, Center

This book could not have reached publication without the encouragement, help and support from a number of colleagues and friends. In particular we would like to thank the following:

Tim Sennitt for his help in preparing the printed material and doing much of the donkey-work to bring this to publication.

Joanne Rearnkham, Barry Bryan and David Jaramillo for their support in enabling access to the materials necessary to produce this book.

Chris Perritt and Glen Brew for making available the original Design Workbook and Functional Specifications for OS/2 2.0.

Charlie Schmitt for his original work on converting the kernel debugger code into a dump formatter.

Jeff Mielke and David Jaramillo for their work on PMDF, the structure compiler and continued work on the dump formatter.

Allen Gilbert for making available documentation on System Trace, which has been reproduced in an edited form in this book. Also, for making available an early version of the dump formatter without which it would not have been possible to develop the original Dump Formatter class.

Doug Azzarito for supplying the material on Kernel Debugger Remote Debug Setup.

James Taylor for providing the basis of the lab exercises relating to PM hangs.

Marie Jazynka, one of the first OS/2 debuggers, for patient encouragement of a great many OS/2 debugging people.

Our management teams, without whose foresight and support none of this work would ever have started. These include:

- Hermann Lamberti General Manager for PSM EMEA; Gordon Bell - director PSM EMEA Technical Marketing; Chris Brown - manager PSM OEM and Enterprise Technical Marketing and Brian Rose - manager PSM Project Office; Roy Aho - Director of the Solution Developer Technical Support Center, for encouraging the beginnings of this several years ago; Terry Gray, manager of Platform Competency and Operation, within Solution Developer Technical Support, Austin.

Finally to Sarah-Jane and Shelly, for supporting many very extended working days and weeks.

Chapter 1. Kernel Debugger User Guide

The Kernel Debugger is essentially a replacement OS/2 Kernel module that contains an in-built debugger component. With the debugger one may halt system execution, inspect and alter memory and registers and display system control blocks. The debugger is controlled from a dump ASCII terminal (the debugging console) which is connected to the machine under test (MUT), either directly or via a modem-modem link, through one of its COMx ports. The debugger supports a comprehensive command set, which is fully described in the Chapter 3, “Kernel Debugger and Dump Formatter Command Reference” on page 71 and in other sections of this book.

The debug kernel is distributed in two forms:

ALLSTRICT
This version of the kernel contains all optional self-diagnostic (otherwise known as strict or assertion checking) code. Besides this functional difference many of the system control blocks have extra accounting and signature fields. This has a number of consequences that may affect problem diagnosis:

1. Performance characteristics will be different since extra checking and accounting is being performed.
2. Memory usage will be different because of extra diagnostic code, extensions to system control blocks and in some cases additional space to cause page faults rather than overlays by erroneous code.
3. Timing critical problems might not be recreatable under the ALLSTRICT kernel.
4. Secondary problems may be detected or even introduced through the use of additional diagnostic code.

HSTRICT
This version of the kernel is essentially the RETAIL kernel with the debugger component. It contains only a limited set of strict checking code. The system control blocks are of the same form as those used by in the RETAIL kernel. The performance characteristics of the HSTRICT kernel are closer to those of the RETAIL kernel than the ALLSTRICT kernel. For this reason the HSTRICT kernel is recommended as a first choice when diagnosing application and non-system problems.

The base version of the ALLSTRICT kernel is distributed with the *OS/2 Developer's Tool kit*. Versions of the HSTRICT and ALLSTRICT kernels for fix packs may be obtained from the following sources:

- The OS/2 Base product CDROM for Warp is distributed with the **ALLSTRICT** kernel and Dump Formatter. (For the initial release of Warp this was only available on the US version of Warp).
- The Developer Connection CDROM - this may be ordered through the Developer Assistance Program (DAP) or the System Library Subscription Service (SLSS).
- From your local IBM Marketing Representative.

Information

Throughout this chapter the term *debugger* is used loosely to mean any of the following where ambiguity is not a problem:

Debug Kernel (HSTRICT or ALLSTRICT).

The debugger component within the debug kernel.

The debugging console.

1.1 Kernel Debugger Local Setup

The following items are required to install and setup a local debug session:

Either the **HSTRICT** or **ALLSTRICT** kernel appropriate to the level of the MUT.

System symbol files. These are optional, but useful breakpoints and system data are difficult to locate without them.

Application symbol files. These are only necessary if you intend to debug complex applications where data and subroutines are difficult to locate without them.

System Trace definition and Formatting files. These are only required if you intend to trace kernel dynamic trace points while using the debug kernel.

A null modem cable.

An asynchronous ASCII dumb terminal or an emulator on another PC. Softerm, which is distributed with OS/2 is suitable. PMDF, which is part of the *OS2PDP* package distributed with this book also provides a terminal emulator interface suitable for use with the Kernel Debugger. Other popular emulators used with the Kernel Debugger include: PMDEBUG, DEBUGO and LOGICOMM.

Confusion sometimes arises over the installation of the kernel debugger, particularly as the OS/2 Developer's Tool kit distributes debug versions of other OS/2 modules. Note in particular:

The debug versions of *OS2LDR*, *PMDD.SYS*, *PMGRE.DLL* and *PMWIN.DLL* are optional. These modules will route additional diagnostic information to the debug console if they are installed.

No modification of CONFIG.SYS is required.

A secondary console attached to the MUT may not be used as a debug console.

1.1.1 Installing the Debug Kernel

If you use the OS/2 Developer's tool kit to install the debug kernel then the installation is performed automatically using the supplied **DBGINST** command. If you choose to install the debug kernel manually then perform the following steps:

1. Copy the debug kernel (OS2KRNLD or OS2KRNLB) to the root directory of the boot drive.
2. Copy the symbol files into the same directories as their corresponding load modules. Usually system symbol files are distributed on a diskettes that have the same directory structure as OS/2 system code. This conveniently

allows the UNPACK command to be used to copy all symbols files in one operation (per diskette).

3. Unhide the RETAIL kernel module using the following command:

   ```
   ATTRIB -r -s -h OS2KRNL
   ```

4. Rename the RETAIL kernel to something unique, for example, OS2KRNLR.

5. Rename the ALLSTRICT or HSTRICT kernel to OS2KRNL. There is no need to hide or make the replaced kernel read-only, unless you wish to protect yourself against accidents!

The MUT is now ready to use in debug mode as soon as it is re-booted. Before that happens the debug console needs to be set up.

Note: It is possible to run the MUT with the debug kernel installed without setting up the debug console. This particularly useful when diagnosing pervasive problems. If the COM port settings are correct when the problem reoccurs then the debug console may be connected at that time.

1.1.2 Debug Terminal Setup

This section describes the connection and setup of the debugging console. You may need to know the operational requirements of both your local COM port (on the MUT) and dumb ASCII terminal. Fortunately the debug kernel does not impose any form of hand-shaking or a fixed COM speed setting. In many cases default settings apply. First we discuss the cable requirements.

A null modem cable is required to connect the MUT to the debug console. This is essentially a 3-wire circuit that connects the two COM connectors together. Some PCs are equipped with a 25-pin sockets, other 9-pin. A null modem cable is a symmetric circuit so we do not distinguish which is the MUT and which the console.

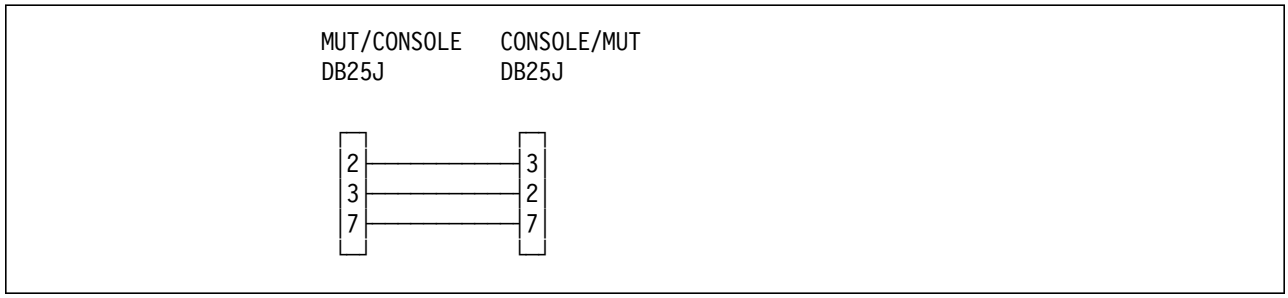


Figure 1. Local 25-to-25 Pin Cable

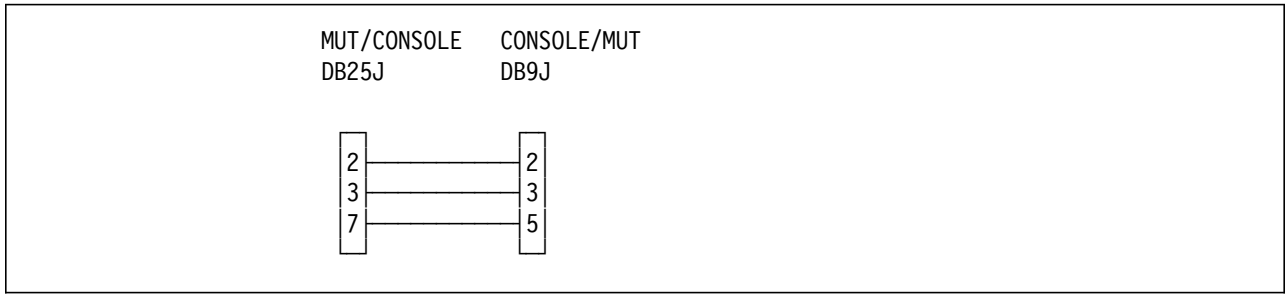


Figure 2. Local 25-to-9 Pin Cable

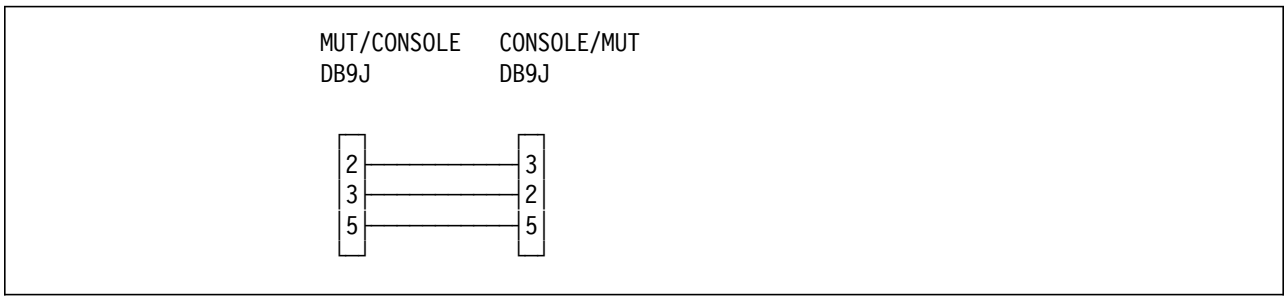


Figure 3. Local 9-to-9 Pin Cable

Note: The three connections involved are:

- RX (receive)
- TX (transmit)
- SG (Signal Ground)

The null modem cable essentially connects RX-TX and SG-SG. The pin conventions for RX and TX on a 25-pin connector reverse those of a 9-pin connect. Thus the 25-9 connection looks like a non-null circuit.

If you intend to debug on a number of different setups then it is worth equipping yourself with the following items, which are commercially available:

- Standard modem cable
- A gender changer
- A null modem convertor
- A 25-9 pin convertor

With these items you should be able to cater for most variations and remote connection as well.

The next thing to consider is the COM port settings. By default the debug kernel will first select COM2. It that is in use then COM1. If you require the debugger to use another COM port, or a non-standard I/O port address then you might need to set this explicitly by using the **.**B command, which should be entered in the KDB.INI initialization file.

By default the kernel debugger initializes the selected COM port to run at 9600 bits per second. If your debugging console requires a different speed setting then you should convey this to the debug kernel using the **.**B command, again entered in the KDB.INI file.

The default communications protocol uses 8 data bits, 1 stop bit and no parity. If this needs to be different, then it may be set using the 0 command also enterd in the KDB.INI file.

Finally some COM ports require the DTR signal to be held high before allowing communication. If this is necessary then it can be set using the debug kernel to write to the I/O port that controls the COM port setup register. This may be done using the the 0 command entered in the KDB.INI file.

Examples of using these commands in KDB.INI is given in the next section.

Having set up the COM port requirements on the MUT, the debug console must be setup to match. Precisely how this is done will depend on whether a dumb terminal or terminal emulator software is used. If you use emulator software under OS/2 you may need to use the OS/2 MODE command to select compatible COM port settings for the debugging console's COM port.

1.1.3 The KDB.INI Initialization File

The debug kernel normally only accepts commands entered at the debugging console. However, during system initialization it will accept commands entered into a text file, which if used, must be called KDB.INI and reside in the root directory of the boot drive.

The KDB.INI file is read after the kernel has loaded and the kernel symbols are loaded and the system is running in protect mode.

Attention

The content of the KDB.INI file is somewhat sensitive. If you make a syntax or format error then you may hang the system and have to re-boot from installation diskettes to recover.

On most systems the use of a KDB.INI file is not required to establish correct operation of the COM port and should be avoided.

Each command must be terminated with a <CR><LF> pair except the last in the file.

The KDB.INI is most easily created using:

```
COPY CON: KDB.INI
```

Enter the commands you require, using the <RETURN> key after each command except the last. For the final command, terminate it using the sequence: Ctrl-Z <RETURN>.

Note: Use of an editor for creating KDB.INI may not be suitable if the <CR><LF> sequence cannot be suppressed from the last line.

The following example hows how to select COM3 at 1200 bps, with DTR held high and to prepare the debugger to intercept any ring 2 or 3 traps.

```
.b 1200t 3e8
O 3ec 1
vsf *
g
```

Notes:

Since the default arithmetic base for the debugger is hexadecimal a **t** suffix is required if the COM port speed is specified in decimal as in the example.

We have assumed a standard port address assignment for COM3, namely 3e8 for data register and 3ec for control register.

The VSF command causes the debugger to intercept all ring2 and ring3 traps and give control to the debug console.

The G command is required unless you want to enter the debugger as soon as the kernel has entered protect mode, loaded its symbol file and executed the KDB.INI file.

1.2 Kernel Debugger Remote Setup

This section describes how to use the kernel debugger remotely, that is with a modem-modem link between the machine under test (MUT) and the debugging console.

The first step is to install the debug kernel and symbols files on the MUT as described preceding section, (see 1.1, “Kernel Debugger Local Setup” on page 2 for more information.)

Although the Debug Kernel will work with nearly any modem; the configuration details are unique to each modem. This topic describes the setup of several modems, and gives general guidelines for setting up others.

1.2.1 Items Required to Setup a System for Remote Debugging

To complete the installation, you will need:

- The RETAIL and either the HSTRICT or ALLSTRICT Kernel
- A modem
- A modem data cable
- An analog dial-in telephone line
- Communications software

1.2.1.1 Modem

Most asynchronous modems currently available will be suitable for use as a remote-debug modem. For best performance, the modem should:

- Support auto-answer operation
- Support locked DTE speed at 9600 bps
- Allow connections at CCITT V.32 (9600 bps), and V.22bis (2400 bps)
- Support error-correction (MNP or V.42)
- Save configuration so a power-outage does not lose settings

1.2.1.2 Modem Data Cable

The configuration of the cable used to connect the modem to the MUT is not important. Any serial data cable should have the connections required by the debug kernel. Just make sure you don′t use a *null-modem* cable. You will either need a 25-to-25 pin cable (for connection to the built-in serial port on a PS/2), or a 25-to-9 pin cable (for connection to a 9-pin serial port).

Required connections for remote debug cable:

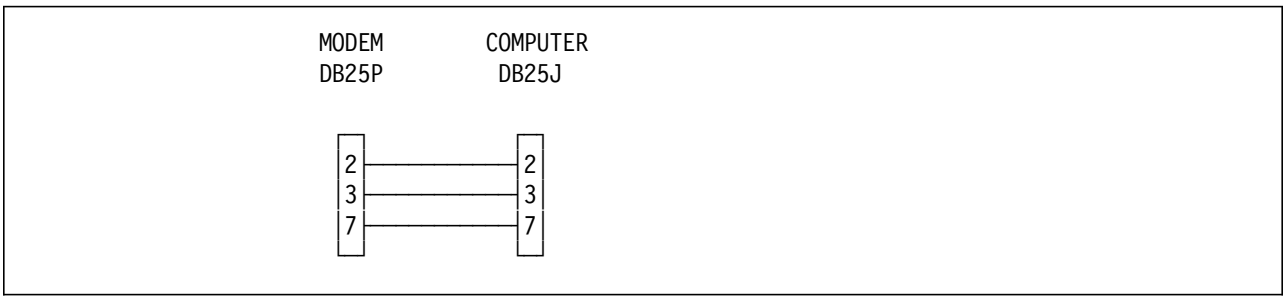


Figure 4. Modem 25-to-25 Pin Cable

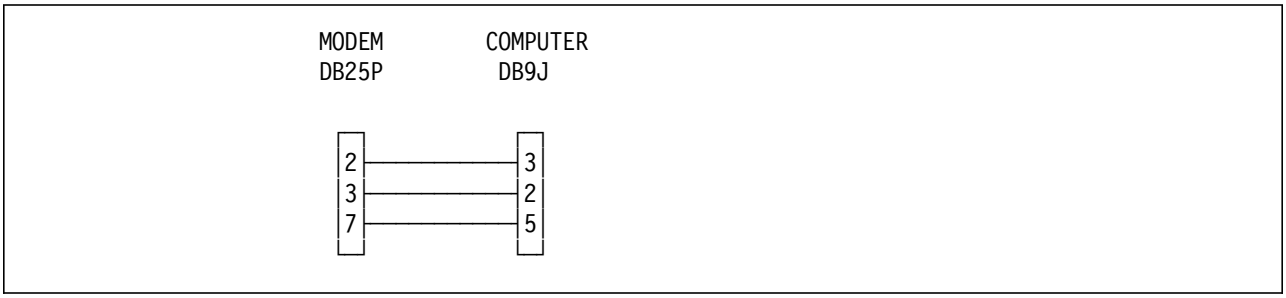


Figure 5. Modem 25-to-9 Pin Cable

Notice the 25-to-9 pin cable reverses pins 2 and 3. Do not confuse this with a null-modem cable - the signals on a 25-to-9 pin cable are normally reversed.

1.2.1.3 Analog Dial-in Telephone Line

In order to call the modem and connect to the MUT, you will need a standard voice-grade telephone line that can be direct-dialed. A connection can be made if the line must go through a switchboard, but it makes it more difficult for the person doing the debugging. Digital telephone lines won't work at all with the modem.

1.2.1.4 Communications Software

Any terminal software that can communicate at 9600 bps will suffice. OS/2 2.0 comes with a program (Softerm Custom) that is adequate. **PMDF**, which is part of the OS2PDP package on the CDROM that accompanies this book, also provides a terminal emulation facility. In addition it provides REXX support that allows Kernel Debugger command sequences to be automated.

1.2.2 The Configuration Process

After you have assembled the required items, follow these steps to prepare the MUT for remote debugging:

1. Connect the Modem to the MUT.

   Connect one end of the data cable to the modem, and the other end to the serial port on the MUT. If the MUT has more than one serial port, connect the cable to the port configured as COM2 (the debug kernel uses COM2 by default). On PS/2 systems, the reference diskette can tell you which port is configured as COM2. Connect the telephone line to the modem, and power the modem on.

2. Program the modem for *debug* operation:

Programming the modem may be a complex process, depending on the type of modem and the intended use. There are two ways to program the modem:

- Quick programming for single debug use
- Full programming for ″permanent″ debug use

The *quick* method is simple, but the modem will not be programmed to recover from loss of power or repeated calls. The *full* method allows the modem to be programmed once, and then used whenever debugging is needed.

The quick programming is performed by the debug kernel itself through use of the KDB.INI file. In addition to containing startup commands for the debugger, KDB.INI can also contain modem initialization strings coded as operands to the Kernel Debugger ? command. For this reason, the modem must be connected and powered on when the MUT is booted, and cannot be powered off until debugging is complete.

The first lines of the KDB.INI may will be COM port selection and parameters if defaults are not suitable, for example:

```
.B 1200t 1
```
(Set debugger for 1200 bps, COM port 1)

Following this are the modem initialization strings, which are unique to each type of modem. The commands in the initialization string must:

- Activate *auto-answer*
- Lock the DTE at 9600 bps
- Activate XON/XOFF flow control
- Ignore the DTR signal (not supplied by the debug kernel)
- Suppress result codes

The remaining lines of the KDB.INI file may contain other debugging commands. The last of these is normally G.

The quick programming strings for several popular modems are as follows.

```
? ″AT&F E0 Q1 &B1 &H2 &I2 &D0 S0=1″
```
US Robotics HST and Dual Standard

```
? ″AT&F2 E0 Q2 &D0 &K4 S0=1″
```
Supra FAX/Modem V.32bis

```
? ″AT&F E0 Q1 &D0 \Q1 S0=1″
```
Intel 14.4EX

An alternative quick technique for entering the Hayes modem initialization commands, which avoids the use of KDB.INI is illustrated by the following example. This example assumes that the default COM2 port is to be used:

1. In CONFIG.SYS add the following line

```
RUN=C:OS2CMD.EXE /K C:MODEM.CMD
```

2. Edit a file called MODEM.CMD and enter the following two lines

```
MODE COM2:9600,N,8,1
COPY MODEM COM2
```

3. Edit a file called MODEM and enter the following line

AT&K4&D0S0=1&W

To use Full programming, you will configure the modem with the same features as in quick programming, but the settings will be stored in the modem′s firmware (or set in modem switches). Determining how to store these settings can be difficult. A thorough study of the modem manual may be required. To program the modem, use a terminal emulation program (for example, the Softerm program that is supplied with OS/2). When programming the modem, set the terminal program for 9600 BPS operation, and type the appropriate modem string. Since the initialization string instructs the modem to suppress result codes, the modem will not return a response. The FULL programming strings for several modems are:

AT&F &B1 &H2 &I2 &W
US Robotics HST and Dual Standard

AT&F2 E0 Q2 &D0 &K4 S0=1 &W
Supra FAX/Modem V.32bis

AT&F E0 Q1 &D0 \Q1 S0=1 &W
Intel 14.4EX

Note: The US Robotics HST Dual Standard does not store all settings, but has external switches instead. After programming the modem, set the switches as follows:

1=ON
(DTR forced ON)

2=don′t care
(result code type)

3=OFF
(result code suppressed)

4=ON
(command echo suppressed)

5=OFF
(auto-answer enabled)

6=do not care
(carrier detect function)

7=ON
(result code in originate mode only)

8=ON
(AT commands enabled)

9=ON
(do not disconnect for +++)

10=OFF
(load NVRAM at power-on)

QUAD=OFF
(normal connect - ON if null modem cable used)

Once the modem is connected, and programmed, the system should be ready for remote debugging. Re-boot the system with the debug kernel installed. When

the telephone rings, the debug modem should answer the phone, and establish connection with the caller. The modem-to-kernel speed should remain at 9600 bps (the default speed used by the debug kernel), but the modem-to-modem speed can be whatever is used by the remote modem. If both modems support error correction, correction will be used.

1.2.2.1 Using Low Speed Modems

If a 9600 bps modem is not available, a slower modem can be used with the debug kernel. If the modem supports *speed conversion* (a 2400 bps modem with error-correction and compression will support speed conversion), setup is straightforward. Construct the proper initialization string for the modem, making sure that the modem's DTE speed (modem-to-debugger) speed is locked at 9600 bps. If the modem does not support speed conversion, construct an initialization string for the modem, and create a KDB.INI file that resets the debugger to the speed supported by the modem. For example, use `.B 2400t 2` for a 2400 bps modem. In this case, the person calling the debugger will have to use the speed supported by the modem.

1.2.2.2 Limitations of this Setup

Since the modem communicates with the MUT at 9600 bps, but can communicate with the remote modem at any speed, the modem must use flow control to avoid data overruns. The only flow control supported by the debug kernel is XON/XOFF. The only problem this causes is when the remote user wants to pause a continuous data display by pressing Ctrl-S. If the modem has also sent a Ctrl-S, the one from the user will be ignored. You may have to press Ctrl-S several times before the display pauses. This is not a problem if the remote user's communications program supports a *scroll-back buffer*, in which case there is no reason to pause the display with Ctrl-S.

1.2.2.3 Troubleshooting

If, after following these directions, you cannot establish a remote debug connection, this guide may help:

Table 1. Modem Troubleshooting Guide

Symptom	Problem	Solution
Modem rings, but doesn't answer.	Modem not set for auto-answer.	Check modem programming, look for AA light on modem.
... " ... " ...	Phone line not connected to modem.	Plug in telephone line to modem.
Modem answers, but no response from debug kernel.	Retail kernel installed.	Remove RETAIL kernel and install DEBUG kernel.
... " ... " ...	Data cable not connected properly.	Connect data cable from modem to MUT. Plug into COM2 if MUT has more than one serial port.
User at the remote modem sees *garbage* on screen, unable to control debug session.	Modem not locked at 9600 bps.	Check modem configuration.
... " ... " ...	Debug Kernel not operating at 9600 bps.	Add `.B 9600T` to KDB.INI file (create file if needed, in root directory of boot drive). Re-boot MUT.

1.3 Controlling the System from the Debugging Console

Having setup the Kernel Debugger for a Local or Remote debug session the system is ready to be controlled from the debugging console. The console is used in two modes, which for convenience we refer to as:

Monitor mode

Command mode

In Monitor mode the console acts merely as an output device for displaying diagnostic messages from the debug kernel and debug versions any other of system modules that write messages to the debugger's COM port. In this mode it is not possible to enter Kernel Debugger commands without having first switched to command mode. In monitor mode the system runs more or less as a retail system except for the performance overheads imparted by the additional diagnostic code.

Monitor mode is in effect initially unless a KDB.INI file is defined.

The console switches to monitor mode after G command is executed.

In Command mode normal system execution is suspended. The debug component of the kernel monitors the debugging console for command input and indicates this with using one of the following command prompts:

> Signifies that the system has been suspended while in real mode.

Signifies that the system has been suspended while in protect mode with paging disabled.

- Signifies that the system has been suspended while in V86 mode with paging disabled.

Signifies that the system has been suspended while in protect mode with paging enabled.

-- Signifies that the system has been suspended while in V86 mode with paging enabled.

In addition to these prompts the Kernel Debugger also uses a data prompt when a command require additional input. This is signified by a single colon prompt ':'. Commands such as R and E may use a data prompt.

Command mode is entered when one of the following events occur:

A fatal exception while executing in ring 0

Any unrecoverable exception occurring in a device driver, file system driver or the OS/2 kernel, will result in a fatal error if it is allowed to be intercepted by the system exception handlers. When this occurs it is usually not possible to restore the system to a running state.

The VTF command may be used to intercept potentially fatal exceptions before the system's exception handlers receive control. If the exception condition is corrected manually, then the system may continue to run after the G command is entered. See Trap and Exception Processing for further information.

An Internal Processing Error (IPE) occurs

Internal processing errors are unrecoverable conditions that are detected by the OS/2 kernel. Some of these are exceptions (described in the previous

bullet); others are inconsistencies that arise from invalid logical conditions or invalid system data. Under the retail kernel IPEs result in the system halting. Under the debug kernel, the console enters command mode after an error message is displayed. IPE messages may be suppressed from displaying as a hard error popups by setting the byte at symbol: *fDebugOnly* to a non-zero value. Under the debug kernel some IPEs are generated for recoverable conditions and allow the system to continue execution after the `G` command is entered. An example of a recoverable IPE is where the loader detects a *bad* or mismatched symbol file for a module it is loading. When this occurs the system displays message:

```
Internal Symbol Error
```

Command mode is entered. If the `G` command is subsequently issued the system will be allowed to continue execution without the *bad* symbol file being activated.

A sticky breakpoint fires

Sticky breakpoints are set using the `BP` and `BR` commands. The system may be returned to a running state after the `G` command is entered.

An unhandled non-maskable interrupt (NMI)

NMIs normally signal hardware error conditions. Under the RETAIL kernel these usually result in TRAP 2 fatal exceptions unless an NMI handler has been registered by a device driver. Under the debug kernel, unhandled NMIs cause control to be given to the debugging console from which it is possible to return the system to a running state using the `G` command.

NMIs are may be generated from several sources, which include:

Channel check

This occurs when an I/O card activates the channel check signal.

Memory parity error

This occurs when memory capable of parity bit generation, detects a parity discrepancy as memory is fetched from RAM.

DMA bus time-out

This occurs when a DMA-driven device uses the bus for longer than the maximum allowed period of 7.8 microseconds.

The watchdog timer interrupt

This occurs when the NMI watchdog (NWD) is enabled and timer interrupts (IRQ 0) are disabled causing loss of timer ticks. OS/2 maintains an NWD count, which If exceeds a maximum value then an IPE is generated. Some hardware/BIOS also maintains an NWD counter, but the precise details of the NWD mechanism are machine specific. For some systems the NWD may not be supported. For further information refer to the appropriate hardware and BIOS reference literature for the machine type under consideration.

Unless the NMI is masked off by setting the mask bit 0x80 in I/O port to 0x70, the NMI channel check provides a means of breaking into the system even when it is disabled for (maskable) interrupts, that is, when the *CLI* instruction has been used to clear the interrupt flag in the *EFLAGS* register. An an ISA-bus system a prototype card may be used to implement the following circuit, which provides an NMI push button switch:

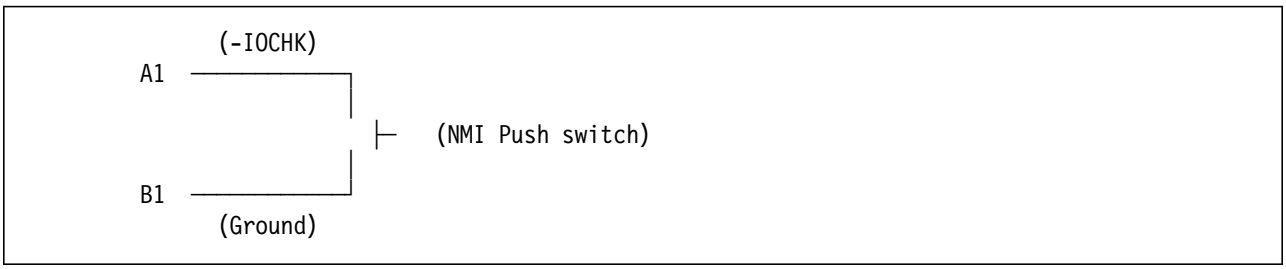


Figure 6. NMI Switch

Note: OS/2 normally only disables NMIs during system initialization and when the Kernel Debugger is running in command mode. However, the Kernel Debugger will allow only one attempt to break in using a channel check NMI, after which NMIs are disabled until the system is re-booted.

An INT 3 instruction is executed

INT 3 instructions are used by the system to implement tracing and software breakpoints. (see the System Trace Facility for additional information). However any program may use INT 3 instructions freely under the Kernel Debugger to cause system execution to be suspended and the debugging console to switch to command mode.

Note: Under the RETAIL kernel, INT 3 instructions other than those implemented by the system for tracing cause code to be terminated with a TRAP 3 exception.

The user enters Ctrl-C from the debugging console.

Unless the system is in a disabled state, the user may type Ctrl-C from the debugging console at any time to cause immediate suspension of normal system execution and the console to switch to command mode.

The user holds down the r-key from the debugging console at system initialization time.

If the *r-key* is held down at system initialization time the debugging console will switch to command mode shortly after the OS2KRNL has entered real-mode for the first time. At this time no symbols have been loaded, paging has never been enabled and the KDB.INI file has not been processed.

Note: In real-mode many of the Kernel Debugger external commands are not available (because they rely on Virtual Memory Management to be initialized). Attempts to use them may cause unpredictable results or even total system failure.

The user holds down the p-key at the debugging console at system initialization time.

If the *p-key* is held down at system initialization time the debugging console will switch to command mode shortly after the OS2KRNL has entered protect-mode for the first time. At this time no symbols have been loaded, paging is disabled and the KDB.INI file has not been processed.

The user holds down the Space-bar from the debugging console at system initialization time.

If the *space-bar* is held down at system initialization time the debugging console will switch to command mode shortly after the OS2KRNL has entered protect-mode and fully initialized. At this time OS2KRNL symbols have been loaded and paging is enabled but the KDB.INI file has not been processed.

The KDB.INI file is processed.

If the KDB.INI file is present then the Kernel Debugger effectively enters command mode by executing Kernel Debugger commands from the KDB.INI file. After the last command is executed, the command prompt appears at the debugging console, unless that last command was a G command.

1.3.1 Controlling Output to the Debugging Console

In both monitor and command mode the following control key sequences are supported:

Ctrl-C

Will cancel the currently running command and return the console to command mode.

Ctrl-S

Will temporarily suspend output to the debugging console and suspend system execution.

Ctrl-Q

Will resume system execution and output to the debugging console.

Note: Ctrl-Q and Ctrl-S correspond to the ASCII asynchronous communications control characters: *XON* and *XOFF*. These may be used by any terminal emulator, which interfaces with the the Kernel Debugger, as a data pacing mechanism.

1.4 Optional System Diagnostic Facilities

Several system components implement optional diagnostic facilities under the debug kernel. These cause additional checking and in some cases detailed information to be displayed at the debugging console when certain debug flags switches are set.

Note: Debugging switches are not a formally architected feature of the OS/2 operating system. They are provided primarily for use by OS/2 developers in debugging and testing the system. They are therefore subject to change or withdrawal without any notice whatsoever.

In this section the following logging facilities are described:

1.4.1 Forcing a System Dump from the Kernel Debugger

Sometimes the situation arises where neither a kernel debug session or a system dump alone are sufficient to analyze a problem. Typically this occurs with problems where evidence of the cause has been removed from the system before the problem occurrence becomes recognized but the problem itself requires lengthy analysis even when the causal conditions are intercepted. Examples of this are problems where:

Storage overlays, may not be noticed until the valid owner of the storage traps at some later time.

A program terminates apparently normally, but unexpectedly.

A deadlock or hang occurs because a resource owner. *forgets* to release ownership of a shared resource.

If the problem is such the there are readily identifiable criteria that allow it to be intercepted closer to its cause, for example by using breakpoints under the Kernel Debugger, then being able to take a dump at such a point can be advantageous.

The simplest technique for initiating a system dump is to type the dump key sequence (*Ctrl-Alt-NumLock-NumLock*) from the keyboard of the system undertest while the debugger is in console mode. Then type the `G` command from the debug console. The keyboard interrupt will be serviced and the stand-alone dump procedure initiated.

In an unattended situation a manually initiated dump may not be feasible. The following techniques discuss how to initiate the system dump in a more automated fashion. In some cases it may be possible to set up the command automation from the KDB.INI initialization file.

The system dump is initiated when the kernel routine RASRST (RAS restart) is called. Normally this occurs from ring 0 when exception management intercepts a trap and TRAPDUMP is coded in the CONFIG.SYS file or when the keyboard device driver (*KDB.SYS*) intercepts a Ctrl-Alt-NumLock-NumLock or Ctrl-Alt-F10-F10 sequence. From ring 3 RASRST is called indirectly via the *Dos32ForceSystemDump* API since RASRST is not addressable from any user code selectors. The Kernel Debugger `G` command allows an address to be specified where execution is to continue from, which provides a means calling the system dump routine from the debugging console. Before using this technique, the following points must be understood:

RASRST is not addressable from user code selectors since they have an upper address boundary of at most 512MB.

RASRST requires to be executed using a 16-bit code selector.

RASRST requires a ring 0 stack selector to be active

Dos32ForceSystemDump requires a 32-bit code selector, such as *5b*.

On some early versions of OS/2 2.1 Dos32ForceSystemDump is unreliable.

The symbol Dos32ForceSystemDump occurs in both DOSCALL1.DLL and the callgate entry point in OS2KRNL.

From ring 0 the following command will generally be successful in initiating a system dump:

```
g =rasrst
```

From ring 2 or ring 3, 32-bit code the following commands will be successful providing Dos32ForceSystemDump is working correctly. The address of DOSCALL1:DOS32FORCESYSTEMDUMP is determined first, then a call to Dos32ForceSystemDump is made:

```
ln dos32forcesystemdump
%1a027c78 doscall1:FLAT32:DOS32FORCESYSTEMDUMP

g =1a027c78
```

For 16-bit application code the CS register must be to to a value that will address DOSCALL1.DOS32FORCESYSTEMDUMP. A suitable selector would be *5b* for ring-3 code and *5a* for ring-2. So, for 16-bit code this procedure becomes:

```
ln dos32forcesystemdump
%1a027c78 doscall1:FLAT32:DOS32FORCESYSTEMDUMP
r cs 5b  (or r cs 5a)

g =1a027c78
```

If TRAPDUMP is in effect then a dump can be forced by causing an immediate trap. The most effective way to achieve this is to set the current SS selector to 0 using the R command. For example:

```
r ss=00
g
```

If you wish to trap an application the very next time it runs in user mode then use .R to determine the user registers and set a breakpoint on CS:EIP in the context of the application's thread slot and specify that SS be set to zero when the breakpoint fires. For example:

```
.p 2d
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
 002d  000b 0002 000b 0001 blk 0200 7b700000 7b8c68fc 7b8acb60 1eb8 14 mrfilepm

##.r 2d
eax=00000000 ebx=00000000 ecx=0000aa37 edx=0000a9ef esi=00090bff edi=00090000
eip=0000272d esp=0000b228 ebp=0009b230 iopl=2 -- -- -- nv up ei ng nz na pe nc
cs=d02f ss=004f ds=a9ef es=be47 fs=150b gs=0000  cr2=1704b000  cr3=001d9000
doscall1:CONFORM16:postDOSSEMWAIT:
002d|d02f:0000272d c9        leave                     ;br0

##bp d02f:272d,"j wo(tasknumber)==2d,'.r;r ss=0;g';g"
##g

eax=00000000 ebx=00000014 ecx=0009a9ef edx=0000a9ef esi=00090bff edi=00090006
eip=0000272d esp=0000b230 ebp=0009b230 iopl=2 -- -- -- nv up ei ng nz na pe nc
cs=d02f ss=004f ds=a9ef es=be47 fs=150b gs=0000  cr2=01550000  cr3=001d9000
d02f:0000272e ca0800         retf    0008
Symbols linked (calc)
DelayHardError SYS3171: 4 string(s):
 Pid 000b  Tid 0001  Slot 002d  HobMte 03be
 C:\OS2TOOLS\MRFILEPM.EXE
c0000005
1a05272d
P1=00000008  P2=6d640000  P3=XXXXXXXX  P4=XXXXXXXX
EAX=00000000  EBX=00000014  ECX=0009a9ef  EDX=0000a9ef
ESI=00090bff  EDI=00090006
DS=a9ef  DSACC=00f3  DSLIM=00000fff
ES=be47  ESACC=00f3  ESLIM=000017f5
FS=150b  FSACC=00f3  FSLIM=00000030
GS=0000  GSACC=****  GSLIM=********
CS:EIP=d02f:0000272d  CSACC=00df  CSLIM=000054a3
```

```
SS:ESP=0000:0000b230  SSACC=****  SSLIM=********
EBP=0009b230  FLG=00002386

DOSCALL1.DLL 0005:0000272d
```

This technique will successfully terminate an application. If TRAPDUMP is set appropriately then a system dump will be taken.

If TRAPDUMP is not correctly set for taking dumps, it may be dynamically modified from the debugging console. Symbol *DumpDevice* specifies the dump partition or drive letter (without the colon) and *DUMP_ON* is a flag byte that take values 0, 1 or 2 to specify whether TRAPDUMP is *OFF*, *ON* or *R0* respectively. Use the `E` command to modify to these fields according to needs. For example, if we wish to set the equivalent of *TRAPDUMP R0,F* after system initialization then the following command sequence would achieve this:

```
e dump_on 2
e dumpdevice "F"
```

When examining a dump taken by calling RASRST, directly or indirectly, using the `G` command then the registers at the time the Kernel Debugger was last entered can be found at label *_RegSA*. The format of this save area is as follows:

Table 2 (Page 1 of 2). Save Area Format

Offset	Register mnemonic
+0	EAX
+4	EBX
+8	EXC
+c	EDX
+10	ESP
+14	EBP
+18	ESI
+1c	EDI
+20	ES
+22	SS
+24	DS
+26	FS
+28	GS
+2a	EIP
+2e	CS
+30	reserved
+34	EFLAGS
+38	MSW
+3c	GTD limit
+3e	GTD base
+42	reserved
+44	IDT limit
+46	IDT base

Table 2 (Page 2 of 2). Save Area Format	
Offset	**Register mnemonic**
+4a	reserved
+4c	LDTR
+4e	TR
+50	CR2
+54	CR3
+58	DR0
+5c	DR1
+60	DR2
+64	DR3
+68	DR4
+6c	DR5
+70	DR6
+74	DR7
+78	reserved
+7c	TR6
+80	TR7

1.4.2 Virtual Memory Management Lock Trace

Virtual Memory Management implements a logging function that records successful attempts to lock and unlock memory pages.

Memory locking and unlocking is implemented by the Memory Management routines: *VMLockMem* and *VMUnlock*. This routine is available directly to all kernel components and indirectly to device drivers through the following:

DevHlp_Lock

DevHlp_Unlock

DevHlp_VMLock

DevHlp_VMUnlock

It is also available to file system drivers through the following:

MFSH_Lock

MFSH_Unlock

The VM lock trace is activated by setting bit 0 of the VM log flag doubleword to 1. The flag doubleword is located at symbol: *_VMLogFlags*. Since no function is currently assigned to the other bit positions so the lock log may be effectively turned on by setting the byte a *_VMLogFlags* to *0xff* as in the following example:

```
e _vmlogflags
%fff0127c  00.
ff
##g
L base fff32  size 2 flags 1  hob 16 hptda 3b9 ret fff3e551
L base 15e0  size 1 flags 4  hob 4a4 hptda 91 ret fff5a93c
L base 3f  size 1 flags 4  hob 188 hptda 91 ret fff5a93c
```

```
U base 15e0  size 1 flags 4  hob 4a4 hptda 91 ret fff5a983
U base 3f  size 1 flags 4  hob 188 hptda 91 ret fff5a983
L base 15e0  size 1 flags 4  hob 4a4 hptda 91 ret fff5a93c
L base 3f  size 1 flags 4  hob 188 hptda 91 ret fff5a93c
U base 15e0  size 1 flags 4  hob 4a4 hptda 91 ret fff5a983
U base 3f  size 1 flags 4  hob 188 hptda 91 ret fff5a983
L base fff35  size 3 flags 1  hob 16 hptda 4a4 ret fff3e551
L base fe79c  size 4 flags 0  hob 3 hptda 380 ret fff49ec6
U base fe79c  size 4 flags 0  hob 3 hptda 380 ret fff3d173
```

The fields displayed in each lock trace entry are formatted from the constituent parts of the corresponding lock handle. They are defined as follows:

L Indicates a lock request

U Indicates an unlock request

base
The virtual page number (that is the high order 5 digits of the address) of the page(s) to be locked or unlocked

size
The number of pages being locked or unlocked

flags
The following bit settings are defined:

Bit Value	Description
0x01	Lock is a long-term
0x02	Verify lock call
0x04	Lock originated from a DevHlp

hob
The hob of the memory object whose pages are being locked or unlocked

hptda
The hptda of the process that requested the memory lock or unlock

ret The return address from **VMLockMem**, that is, the address of the caller

Note:

The return address is unfortunately of limited use since most calls to *VMLockMem* are made via a limited number of interface routines. In particular, *DevHlp* lock requests are made via *dhw_VMLock* and *SegLockDM*. Unless one can trace in addition the SS:ESP on entry to *VMLock*, the lock trace alone will be insufficient to solve memory locking problem. One possible way of providing more information is to supplement the lock trace with following breakpoint commands:

```
##bp _vmunlock+1,"k ss:sp;g"
##bp _vmlockmem+1,"k ss:sp;g"
##g
0170:fff3e551 fff32d68 00001281 10000000 ffe0068f CodeLockProc + 7c
L base fff32  size 2 flags 1  hob 16 hptda 3b9 ret fff3e551
0170:fff5a93c 015f0000 0000000e 40000000 fe7958c6 _dhw_VMLock + dc
0170:fff3db40 40000000 015f0000 0000000e fe7958c6
0170:00000155 01550000 62d61a84 00000003 5ab40000
L base 15f0  size 1 flags 4  hob 5de hptda 91 ret fff5a93c
```

```
0170:fff5a93c 0003f198 0000000b 40000000 fe795be0 _dhw_VMLock + dc
0170:fff3db40 40000000 0003f198 0000000b fe795be0
0170:00000055 00550000 62d61a84 00000003 5ab40000
L base 3f  size 1 flags 4  hob 196 hptda 91 ret fff5a93c
0170:fff5a983 fe7958c6 00002796 083082fc fe7958c6 _dhw_VMUnlock + 3a
0170:fff3db4c fe7958c6 00001af0 00001100 00000056
0170:00000003 5b030000 08300000 f2a40000 08489254
U base 15f0  size 1 flags 4  hob 5de hptda 91 ret fff5a983
0170:fff5a983 fe795be0 000008c6 0830823f fe795be0 _dhw_VMUnlock + 3a
0170:fff3db4c fe795be0 00001af0 00001100 0000ff56
0170:00000003 5b030000 08300000 f2a40000 08489254
U base 3f  size 1 flags 4  hob 196 hptda 91 ret fff5a983
0170:fff5a93c 015f0000 00000dd6 40000000 fe7958c6 _dhw_VMLock + dc
0170:fff3db40 40000000 015f0000 00000dd6 fe7958c6
0170:00000155 01550000 62d61a84 00000003 5ab40000
L base 15f0  size 1 flags 4  hob 5de hptda 91 ret fff5a93c
0170:fff5a93c 0003f198 0000000b 40000000 fe795be0 _dhw_VMLock + dc
0170:fff3db40 40000000 0003f198 0000000b fe795be0
0170:00000055 00550000 62d61a84 00000003 5ab40000
L base 3f  size 1 flags 4  hob 196 hptda 91 ret fff5a93c
0170:fff5a983 fe7958c6 00002796 083082fc fe7958c6 _dhw_VMUnlock + 3a
0170:fff3db4c fe7958c6 00001af0 00001100 0000ff56
0170:00000003 5b030000 08300000 f2a40000 08489254
U base 15f0  size 1 flags 4  hob 5de hptda 91 ret fff5a983
0170:fff5a983 fe795be0 000008c6 0830823f fe795be0 _dhw_VMUnlock + 3a
0170:fff3db4c fe795be0 00001af0 00001100 0000ff56
0170:00000003 5b030000 08300000 f2a40000 08489254
U base 3f  size 1 flags 4  hob 196 hptda 91 ret fff5a983
0170:fff3e551 fff351dc 000022f5 10000000 ffe0053f CodeLockProc + 7c
L base fff35  size 3 flags 1  hob 16 hptda 3b9 ret fff3e551
0170:fff4a218 ffe0053f ffe0052b 00082006 00000000 _CodeLockHook + 2c
0170:fff42df7 ffffffff ffffffff 7b71ff40 7b71ff40 KMDispatchHook + a3
U base fff35  size 3 flags 1  hob 16 hptda 3b9 ret fff4a218
0170:fff4a218 ffe0068f ffe0067b 00082006 00000006 _CodeLockHook + 2c
0170:fff42df7 ffffffff ffffffff 7b71ff40 7b71ff40 KMDispatchHook + a3
U base fff32  size 2 flags 1  hob 16 hptda 3b9 ret fff4a218
```

Given that the K command rapidly looses synchronization with the correct stack frame pointer one may have to resort to using:

```
##bp _vmunlock+1,"dw ss:sp l80;g"
##bp _vmlockmem+1,"dw ss:sp l80;g"
```

Refer to the Kernel Debugger K command and BP command for further information.

Related information on memory locking may be found under the description of the Kernel Debugger .MO command.

The latest versions of OS/2 2.11 and OS/2 3.0 have implemented a new Kernel Debugger command that facilitates an alternative method for analyzing memory locking problems. See the Kernel Debugger .MK command for details.

1.4.3 Virtual Memory Management System Heap Validation

The system will perform additional validation of the kernel heap structures under the debug kernel if the byte at label *_vmkhGflags* is set to a non-zero value.

There is a noticeable performance overhead when this option is activated. Therefore it is recommended that it is only used when a heap corruption problem is suspected.

The system will validate the linkages between various heap structures. If an error is detected, then an IPE is generated with one of the following messages:

VMKSH: Invalid hint pointers

VMKSH: Invalid number of ksh descriptors

VMKSH: Invalid number of ksh blocks

Invalid heap block header at address: ssss:oooooooo

Preceding block at address: ssss:oooooooo

No preceding block

1.4.4 System Loader Logging Facility

The system loader provides optional logging and checking under the debug kernel. These optional facilities may be activated selectively by setting bits in the *_LdrDebugFlags* flags doubleword as follows:

Note: The flags described are those implemented in OS/2 Warp V3.0. Slightly different, but similar messages are generated for earlier releases of OS/2.

0x00000001

This will cause the loader to break into the debugger using an INT 3 instruction if any of the following error conditions are detected:

Not enough memory

Caching error

Invalid Ordinal

Procedure not found

Bad EXE format

Invalid segment number

Invalid CALLGATE

Network Disconnected

---

[1] The term loader applies to two distinct components under OS/2:

OS2LDR This is the OS2KRNL loader. One of its functions is to load the OS2KRNL module at boot time. After the system has booted OS2LDR provides the CBIOS layer for the kernel.

System Loader This is a component of the kernel. It is responsible for loading program modules, DLLs, Device Drivers and File System Drivers.

The logging facility discussed in this section applies to the System Loader.

0x00000002

This will generate log entries when *LDRGetPage* exits with a non-zero return code. *LDRGetPage* is called to demand load a page within a object of a load module. The message logged is of the following form:

```
ldrGP bad cr2=nnnnnnnn rc=mmmmmmmm
```

cr2= is the page fault address and **rc=** is the *LDRGetPage* return code.

0x00000004

This generates log entries when *LDRGetPage* is called to demand load a page within a object of a load module. The message logged is of the form:

```
ldrGP cr2=nnnnnnnn hMTE=hhhh bno=oo
      name=pppppppppppppppp
```

cr2= Is the page fault address.

hMTE= Is the module's hmte.

bno= Is page number within the module.

name= is the module's full name taken from the SMTE.

0x00000018

This switch causes log information to be generated when DLL modules are loaded and initialized. The following messages are logged:

```
ldrDLM entry - slot ssss ptda pppppppp

ldrDLM name  - slot ssss name nnnnnnnn

ldrDLM free  - slot ssss

ldrDLM exit  - slot ssss

tk SD has-init slot=ssss

tk SD no-init slot=ssss

tk SD pre-inc slot=ssss cnest=nnnn

tk IN pre-dec slot=ssss cnest=nnnn

tk LIn slot=ssss cnest=nnnn
```

slot Is the thread slot in which the DLL is being processed,

ptda Is the address of the PTDA for this slot

name Is the DLL module name

cnest Nesting counter for **TKLibiStartDispatch**

ldrDLM entry Marks entry to *w_loadmodule*, the *DosLoadModule* worker routine.

ldrDLM name Marks the successful request for the DLL initialization mutex semaphore (&ptdadlmsem.).

ldrDLM free Marks the release of the mutex semaphore. *Exit* marks the exiting of *w_loadmodule*.

ldrDLM exit Marks the exit from *w_loadmodule*.

tk SD Marks events in *TKLibiStartDispatch*.

tk IN and tk LIn Mark events in *TKLiniInitNextDLL*

0x00000080

This switch requests that import initialization be recorded. Messages of the following format are generated:

```
lpi, Recording init hMTE=hhhh, flags1=ffffffff, name=nnnnnnnnn

lpi, Skipping init hMTE=hhhh, flags1=ffffffff, name=nnnnnnnnn

lpi, Processing imports slot=ssss, module=nnnnnnnnn

lrm, Recording init hMTE=hhhh, flags1=ffffffff, name=nnnnnnnn

lrm, Skipping init hMTE=hhhh, flags1=ffffffff, name=nnnnnnnnn
```

hMTE Is the module handle

flags1 Are the flags MTE flags field. (See the **.LM** command for details).

name Is the full module name taken from the SMTE.

module Is the full module name taken from the SMTE.

lpi, Recording init
Logs the processing of system DLL imports from the system DLL names table in EXE file loading.

lpi, Skipping init
Logs system DLL names not imported in EXE file loading.

lpi, Processing imports
Logs the processing of DLL initialization as the result of imports being present in an EXE module.

lrm, Recording init
Logs imported DLL initialization being recorded.

lrm, Skipping init
Logs imported DLLs skipping initialization.

0x00000100

Logs when the loader cannot load an object at the compiler/linker designated base address. The message logged appears as follows:

```
Cannot load nnnnnnnn at the requested base address
```

where *nnnnnnnn* is the module name.

0x00000800

Logs the processing of the DLL import tree. The following messages appear:

```
lpi, Processing imports slot=ssss, module=nnnnnnnnn

ldr walking tree hMTE=hhhh, name=nnnnnnnn

ldr walking tree going down

ldr walking tree going up
```

lpi, Processing imports
marks the initiation of the process for slot ***ssss*** and module ***nnnnnnnn***.

ldr walking tree hMTE=hhhh, name=nnnnnnnn
marks the processing of an imported DLL, whose handle is ***hhhh*** and name is ***nnnnnnnn***

ldr walking tree going up
marks a backward progression through the import tree.

ldr walking tree going down
marks a forward progression through the import tree.

1.4.4.1 Example loader log

The following is an example of a loader log where all logging options have been activated. This illustrates the loader activity recorded when the **FAXWORK.EXE** icon was clicked on:

```
lpi Processing imports slot=0022, module=H:\FAXWORKS\FAXWORKS.EXE
ldr walking tree hMTE=05e1, name=H:\FAXWORKS\FAXWORKS.EXE
ldr walking tree going down
ldr walking tree hMTE=029a, name=H:\OS2\DLL\PMWIN.DLL
ldr walking tree going down
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0293, name=H:\OS2\DLL\PMGPI.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=029b, name=H:\OS2\DLL\MOUCALLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=01e5, name=H:\OS2\DLL\VIOCALLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0262, name=H:\OS2\DLL\NLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=029c, name=H:\OS2\DLL\PMSHAPI.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0111, name=H:\OS2\DLL\SESMGR.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going up
ldr walking tree hMTE=029a, name=H:\OS2\DLL\PMWIN.DLL
ldr walking tree going up
ldr walking tree hMTE=05e1, name=H:\FAXWORKS\FAXWORKS.EXE
ldr walking tree going down
ldr walking tree hMTE=02b8, name=H:\OS2\DLL\PMSPL.DLL
ldr walking tree going down
ldr walking tree hMTE=0104, name=H:\OS2\DLL\MSG.DLL
ldr walking tree going up
ldr walking tree hMTE=02b8, name=H:\OS2\DLL\PMSPL.DLL
ldr walking tree going down
ldr walking tree hMTE=02be, name=H:\OS2\DLL\SPL1B.DLL
ldr walking tree going up
ldr walking tree hMTE=02b8, name=H:\OS2\DLL\PMSPL.DLL
```

```
ldr walking tree going up
ldr walking tree hMTE=05e1, name=H:\FAXWORKS\FAXWORKS.EXE
ldr walking tree going down
ldr walking tree hMTE=0419, name=H:\OS2\DLL\HELPMGR.DLL
ldr walking tree going up
ldr walking tree hMTE=05e1, name=H:\FAXWORKS\FAXWORKS.EXE
ldr walking tree going down
ldr walking tree hMTE=02ac, name=H:\OS2\DLL\PMDRAG.DLL
ldr walking tree going down
ldr walking tree hMTE=02a3, name=H:\OS2\DLL\PMCTLS.DLL
ldr walking tree going up
ldr walking tree hMTE=02ac, name=H:\OS2\DLL\PMDRAG.DLL
ldr walking tree going up
ldr walking tree hMTE=05e1, name=H:\FAXWORKS\FAXWORKS.EXE
ldr walking tree going down
ldr walking tree hMTE=0279, name=H:\OS2\DLL\PMWP.DLL
ldr walking tree going down
ldr walking tree hMTE=029d, name=H:\OS2\DLL\IMP.DLL
ldr walking tree going up
ldr walking tree hMTE=0279, name=H:\OS2\DLL\PMWP.DLL
ldr walking tree going down
ldr walking tree hMTE=02a8, name=H:\OS2\DLL\SEAMLESS.DLL
ldr walking tree going down
ldr walking tree hMTE=02b1, name=H:\OS2\DLL\PMVIOP.DLL
ldr walking tree going up
ldr walking tree hMTE=02a8, name=H:\OS2\DLL\SEAMLESS.DLL
ldr walking tree going up
ldr walking tree hMTE=0279, name=H:\OS2\DLL\PMWP.DLL
ldr walking tree going down
ldr walking tree hMTE=02ad, name=H:\OS2\DLL\SOM.DLL
ldr walking tree going up
ldr walking tree hMTE=0279, name=H:\OS2\DLL\PMWP.DLL
ldr walking tree going up
ldr walking tree hMTE=05e1, name=H:\FAXWORKS\FAXWORKS.EXE
ldr walking tree going up
lrm, Skipping init hMTE=05e1, flags1=20903150, name=H:\FAXWORKS\FAXWORKS.EXE
lrm, Skipping init hMTE=0279, flags1=e498b394, name=H:\OS2\DLL\PMWP.DLL
lrm, Skipping init hMTE=02ad, flags1=e498b396, name=H:\OS2\DLL\SOM.DLL
lrm, Recording init hMTE=02a8, flags1=e098b395, name=H:\OS2\DLL\SEAMLESS.DLL
lrm, Skipping init hMTE=02b1, flags1=a498b395, name=H:\OS2\DLL\PMVIOP.DLL
lrm, Skipping init hMTE=029d, flags1=2098b398, name=H:\OS2\DLL\IMP.DLL
lrm, Skipping init hMTE=02ac, flags1=a498b388, name=H:\OS2\DLL\PMDRAG.DLL
lrm, Skipping init hMTE=02a3, flags1=e498b394, name=H:\OS2\DLL\PMCTLS.DLL
lrm, Skipping init hMTE=0419, flags1=a098b39a, name=H:\OS2\DLL\HELPMGR.DLL
lrm, Skipping init hMTE=02b8, flags1=e498b394, name=H:\OS2\DLL\PMSPL.DLL
lrm, Skipping init hMTE=02be, flags1=a098b398, name=H:\OS2\DLL\SPL1B.DLL
lrm, Skipping init hMTE=0104, flags1=2098b388, name=H:\OS2\DLL\MSG.DLL
lrm, Skipping init hMTE=029a, flags1=2098b388, name=H:\OS2\DLL\PMWIN.DLL
lrm, Skipping init hMTE=0281, flags1=e498b394, name=H:\OS2\DLL\PMMERGE.DLL
lrm, Skipping init hMTE=0111, flags1=2098b388, name=H:\OS2\DLL\SESMGR.DLL
lrm, Skipping init hMTE=029c, flags1=2098b388, name=H:\OS2\DLL\PMSHAPI.DLL
lrm, Skipping init hMTE=0262, flags1=2098b388, name=H:\OS2\DLL\NLS.DLL
lrm, Skipping init hMTE=01e5, flags1=2098b388, name=H:\OS2\DLL\VIOCALLS.DLL
lrm, Skipping init hMTE=029b, flags1=2098b388, name=H:\OS2\DLL\MOUCALLS.DLL
lrm, Recording init hMTE=0293, flags1=e498b394, name=H:\OS2\DLL\PMGPI.DLL
lpi, Recording init hMTE=0279, flags1=e498b394, name=H:\OS2\DLL\PMWP.DLL
lpi, Recording init hMTE=02ad, flags1=e498b396, name=H:\OS2\DLL\SOM.DLL
lpi, Recording init hMTE=02b1, flags1=a498b395, name=H:\OS2\DLL\PMVIOP.DLL
lpi, Recording init hMTE=02a3, flags1=e498b394, name=H:\OS2\DLL\PMCTLS.DLL
```

```
lpi, Recording init hMTE=02ac, flags1=a498b388, name=H:\OS2\DLL\PMDRAG.DLL
lpi, Recording init hMTE=02b8, flags1=e498b394, name=H:\OS2\DLL\PMSPL.DLL
lpi, Recording init hMTE=0281, flags1=e498b394, name=H:\OS2\DLL\PMMERGE.DLL
lpi, Recording init hMTE=00f2, flags1=8498b794, name=H:\OS2\DLL\DOSCALL1.DLL
ldrGP cr2=ffe3b000 hMTE=5e1 bno=85
      name = H:\FAXWORKS\FAXWORKS.EXE
tk SD has-init slot=22
ldrGP cr2=13fa0000 hMTE=f2 bno=2b
      name = H:\OS2\DLL\DOSCALL1.DLL
ldrGP cr2=13fa1000 hMTE=f2 bno=2c
      name = H:\OS2\DLL\DOSCALL1.DLL
ldrGP cr2=13fc0000 hMTE=f2 bno=2e
      name = H:\OS2\DLL\DOSCALL1.DLL
ldrGP cr2=13e30000 hMTE=281 bno=106
      name = H:\OS2\DLL\PMMERGE.DLL
ldrDLM entry - slot 36 ptda ab99a000
ldrDLM name  - slot 36 name H:\OS2\DLL\PMATM.DLL
lpi Processing imports slot=0036, module=H:\OS2\DLL\PMATM.DLL
ldr walking tree hMTE=0354, name=H:\OS2\DLL\PMATM.DLL
ldr walking tree going down
ldr walking tree hMTE=029c, name=H:\OS2\DLL\PMSHAPI.DLL
ldr walking tree going down
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0293, name=H:\OS2\DLL\PMGPI.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=029a, name=H:\OS2\DLL\PMWIN.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=029b, name=H:\OS2\DLL\MOUCALLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=01e5, name=H:\OS2\DLL\VIOCALLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0262, name=H:\OS2\DLL\NLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0111, name=H:\OS2\DLL\SESMGR.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going up
ldr walking tree hMTE=029c, name=H:\OS2\DLL\PMSHAPI.DLL
ldr walking tree going up
ldr walking tree hMTE=0354, name=H:\OS2\DLL\PMATM.DLL
ldr walking tree going up
lrm, Recording init hMTE=0354, flags1=6498b3c5, name=H:\OS2\DLL\PMATM.DLL
lrm, Skipping init hMTE=029c, flags1=2098b388, name=H:\OS2\DLL\PMSHAPI.DLL
tk SD has-init slot=36
tk SD pre-inc slot=36 cnest=1
ldrDLM free  - slot 36
ldrDLM exit  - slot 36
tk LIn slot=36 cnest=1
```

```
ldrDLM entry - slot 36 ptda ab99a000
ldrDLM name  - slot 36 name DISPLAY
lpi Processing imports slot=0036, module=H:\OS2\DLL\DISPLAY.DLL
ldr walking tree hMTE=034b, name=H:\OS2\DLL\DISPLAY.DLL
ldr walking tree going down
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0293, name=H:\OS2\DLL\PMGPI.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=029a, name=H:\OS2\DLL\PMWIN.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=029b, name=H:\OS2\DLL\MOUCALLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=01e5, name=H:\OS2\DLL\VIOCALLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0262, name=H:\OS2\DLL\NLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=029c, name=H:\OS2\DLL\PMSHAPI.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0111, name=H:\OS2\DLL\SESMGR.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going up
ldr walking tree hMTE=034b, name=H:\OS2\DLL\DISPLAY.DLL
ldr walking tree going up
lrm, Recording init hMTE=034b, flags1=2498b394, name=H:\OS2\DLL\DISPLAY.DLL
tk SD has-init slot=36
tk SD pre-inc slot=36 cnest=1
ldrDLM free  - slot 36
ldrDLM exit  - slot 36
ldrGP cr2=13ef0000 hMTE=34b bno=6
      name = H:\OS2\DLL\DISPLAY.DLL
tk LIn slot=36 cnest=1
ldrDLM entry - slot 36 ptda ab99a000
ldrDLM name  - slot 36 name IBMS332
lpi Processing imports slot=0036, module=H:\OS2\DLL\IBMS332.DLL
ldr walking tree hMTE=0362, name=H:\OS2\DLL\IBMS332.DLL
ldr walking tree going down
ldr walking tree hMTE=0368, name=H:\OS2\DLL\PMGRE.DLL
ldr walking tree going down
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0293, name=H:\OS2\DLL\PMGPI.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=029a, name=H:\OS2\DLL\PMWIN.DLL
```

```
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=029b, name=H:\OS2\DLL\MOUCALLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=01e5, name=H:\OS2\DLL\VIOCALLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0262, name=H:\OS2\DLL\NLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=029c, name=H:\OS2\DLL\PMSHAPI.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0111, name=H:\OS2\DLL\SESMGR.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going up
ldr walking tree hMTE=0368, name=H:\OS2\DLL\PMGRE.DLL
ldr walking tree going up
ldr walking tree hMTE=0362, name=H:\OS2\DLL\IBMS332.DLL
ldr walking tree going up
lrm, Skipping init hMTE=0362, flags1=2098b398, name=H:\OS2\DLL\IBMS332.DLL
lrm, Skipping init hMTE=0368, flags1=2098b388, name=H:\OS2\DLL\PMGRE.DLL
tk SD no-init slot=36
ldrDLM free  - slot 36
ldrDLM exit  - slot 36
ldrDLM entry - slot 36 ptda ab99a000
ldrDLM name  - slot 36 name H:\OS2\DLL\IBMS332.DLL
tk SD no-init slot=36
ldrDLM free  - slot 36
ldrDLM exit  - slot 36
ldrDLM entry - slot 36 ptda ab99a000
ldrDLM name  - slot 36 name dspres
lpi Processing imports slot=0036, module=H:\OS2\DLL\DSPRES.DLL
ldr walking tree hMTE=036a, name=H:\OS2\DLL\DSPRES.DLL
ldr walking tree going up
lrm, Skipping init hMTE=036a, flags1=2098b388, name=H:\OS2\DLL\DSPRES.DLL
tk SD no-init slot=36
ldrDLM free  - slot 36
ldrDLM exit  - slot 36
ldrDLM entry - slot 36 ptda ab99a000
ldrDLM name  - slot 36 name H:\OS2\DLL\COMETDLL.DLL
lpi Processing imports slot=0036, module=H:\OS2\DLL\COMETDLL.DLL
ldr walking tree hMTE=037e, name=H:\OS2\DLL\COMETDLL.DLL
ldr walking tree going down
ldr walking tree hMTE=0368, name=H:\OS2\DLL\PMGRE.DLL
ldr walking tree going down
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0293, name=H:\OS2\DLL\PMGPI.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
```

```
ldr walking tree hMTE=029a, name=H:\OS2\DLL\PMWIN.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=029b, name=H:\OS2\DLL\MOUCALLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=01e5, name=H:\OS2\DLL\VIOCALLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0262, name=H:\OS2\DLL\NLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=029c, name=H:\OS2\DLL\PMSHAPI.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0111, name=H:\OS2\DLL\SESMGR.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going up
ldr walking tree hMTE=0368, name=H:\OS2\DLL\PMGRE.DLL
ldr walking tree going up
ldr walking tree hMTE=037e, name=H:\OS2\DLL\COMETDLL.DLL
ldr walking tree going down
ldr walking tree hMTE=0104, name=H:\OS2\DLL\MSG.DLL
ldr walking tree going up
ldr walking tree hMTE=037e, name=H:\OS2\DLL\COMETDLL.DLL
ldr walking tree going up
lrm, Recording init hMTE=037e, flags1=e098b396, name=H:\OS2\DLL\COMETDLL.DLL
lrm, Skipping init hMTE=0104, flags1=2098b388, name=H:\OS2\DLL\MSG.DLL
lrm, Skipping init hMTE=0368, flags1=2098b388, name=H:\OS2\DLL\PMGRE.DLL
lrm, Skipping init hMTE=029a, flags1=2098b388, name=H:\OS2\DLL\PMWIN.DLL
tk SD has-init slot=36
tk SD pre-inc slot=36 cnest=1
ldrDLM free  - slot 36
ldrDLM exit  - slot 36
ldrGP cr2=13b30000 hMTE=37e bno=7
      name = H:\OS2\DLL\COMETDLL.DLL
tk LIn slot=36 cnest=1
ldrDLM entry - slot 36 ptda ab99a000
ldrDLM name  - slot 36 name PMSPL
tk SD no-init slot=36
ldrDLM free  - slot 36
ldrDLM exit  - slot 36
ldrGP cr2=13d90000 hMTE=2b8 bno=31
      name = H:\OS2\DLL\PMSPL.DLL
ldrGP cr2=13940000 hMTE=2a3 bno=7b
      name = H:\OS2\DLL\PMCTLS.DLL
ldrDLM entry - slot 36 ptda ab99a000
ldrDLM name  - slot 36 name PMSDMRI
lpi Processing imports slot=0036, module=H:\OS2\DLL\PMSDMRI.DLL
ldr walking tree hMTE=02c6, name=H:\OS2\DLL\PMSDMRI.DLL
ldr walking tree going up
lrm, Skipping init hMTE=02c6, flags1=2098b388, name=H:\OS2\DLL\PMSDMRI.DLL
tk SD no-init slot=36
```

```
ldrDLM free  - slot 36
ldrDLM exit  - slot 36
ldrGP cr2=13c65000 hMTE=2ad bno=1c
      name = H:\OS2\DLL\SOM.DLL
ldrGP cr2=13f60000 hMTE=279 bno=bf
      name = H:\OS2\DLL\PMWP.DLL
ldrGP cr2=13f40000 hMTE=279 bno=ba
      name = H:\OS2\DLL\PMWP.DLL
ldrGP cr2=13f7d000 hMTE=279 bno=cf
      name = H:\OS2\DLL\PMWP.DLL
tk LIn slot=36 cnest=0
ldrGP cr2=47000 hMTE=5e1 bno=38
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=e6000 hMTE=5e1 bno=55
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=54000 hMTE=5e1 bno=45
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=4a000 hMTE=5e1 bno=3b
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=ec000 hMTE=5e1 bno=5b
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=f0000 hMTE=5e1 bno=5f
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=48000 hMTE=5e1 bno=39
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=52000 hMTE=5e1 bno=43
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=e8000 hMTE=5e1 bno=57
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=f8000 hMTE=5e1 bno=67
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=e7000 hMTE=5e1 bno=56
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=f1000 hMTE=5e1 bno=60
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=ee000 hMTE=5e1 bno=5d
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=49000 hMTE=5e1 bno=3a
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=ed000 hMTE=5e1 bno=5c
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=13000 hMTE=5e1 bno=4
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=115000 hMTE=5e1 bno=84
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=32000 hMTE=5e1 bno=23
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=e1000 hMTE=5e1 bno=50
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=33000 hMTE=5e1 bno=24
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=4c000 hMTE=5e1 bno=3d
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=4b000 hMTE=5e1 bno=3c
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrDLM entry - slot 36 ptda ab99a000
ldrDLM name  - slot 36 name H:\FAXWORKS\FX044.LOL
lpi Processing imports slot=0036, module=H:\FAXWORKS\FX044.LOL
ldr walking tree hMTE=060b, name=H:\FAXWORKS\FX044.LOL
```

```
ldr walking tree going up
lrm, Skipping init hMTE=060b, flags1=2098b1c8, name=H:\FAXWORKS\FX044.LOL
ldrDLM free  - slot 36
ldrDLM exit  - slot 36
ldrDLM entry - slot 36 ptda ab99a000
ldrDLM name  - slot 36 name SND
lpi Processing imports slot=0036, module=H:\MMOS2\DLL\SND.DLL
ldr walking tree hMTE=00fe, name=H:\MMOS2\DLL\SND.DLL
ldr walking tree going down
ldr walking tree hMTE=029a, name=H:\OS2\DLL\PMWIN.DLL
ldr walking tree going down
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0293, name=H:\OS2\DLL\PMGPI.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=029b, name=H:\OS2\DLL\MOUCALLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=01e5, name=H:\OS2\DLL\VIOCALLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0262, name=H:\OS2\DLL\NLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=029c, name=H:\OS2\DLL\PMSHAPI.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0111, name=H:\OS2\DLL\SESMGR.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going up
ldr walking tree hMTE=029a, name=H:\OS2\DLL\PMWIN.DLL
ldr walking tree going up
ldr walking tree hMTE=00fe, name=H:\MMOS2\DLL\SND.DLL
ldr walking tree going down
ldr walking tree hMTE=0104, name=H:\OS2\DLL\MSG.DLL
ldr walking tree going up
ldr walking tree hMTE=00fe, name=H:\MMOS2\DLL\SND.DLL
ldr walking tree going up
lrm, Recording init hMTE=00fe, flags1=6098b396, name=H:\MMOS2\DLL\SND.DLL
lrm, Skipping init hMTE=0104, flags1=2098b388, name=H:\OS2\DLL\MSG.DLL
lrm, Skipping init hMTE=029a, flags1=2098b388, name=H:\OS2\DLL\PMWIN.DLL
lrm, Skipping init hMTE=029c, flags1=2098b388, name=H:\OS2\DLL\PMSHAPI.DLL
lrm, Skipping init hMTE=0262, flags1=2098b388, name=H:\OS2\DLL\NLS.DLL
tk SD has-init slot=36
tk SD pre-inc slot=36 cnest=1
ldrDLM free  - slot 36
ldrDLM exit  - slot 36
ldrGP cr2=13310000 hMTE=fe bno=12
      name = H:\MMOS2\DLL\SND.DLL
ldrGP cr2=13311000 hMTE=fe bno=13
      name = H:\MMOS2\DLL\SND.DLL
tk LIn slot=36 cnest=1
```

```
ldrDLM entry - slot 36 ptda ab99a000
ldrDLM name  - slot 36 name PMCTLS
ldrDLM free  - slot 36
ldrDLM exit  - slot 36
ldrGP cr2=45000 hMTE=5e1 bno=36
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=4e000 hMTE=5e1 bno=3f
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178f5000 hMTE=60b bno=6
      name = H:\FAXWORKS\FX044.LOL
ldrGP cr2=e2000 hMTE=5e1 bno=51
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178f6000 hMTE=60b bno=7
      name = H:\FAXWORKS\FX044.LOL
ldrGP cr2=178f5000 hMTE=60b bno=6
      name = H:\FAXWORKS\FX044.LOL
ldrGP cr2=80000 hMTE=5e1 bno=49
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=3c000 hMTE=5e1 bno=2d
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=39000 hMTE=5e1 bno=2a
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=fc000 hMTE=5e1 bno=6b
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=f9000 hMTE=5e1 bno=68
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=fa000 hMTE=5e1 bno=69
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=e4000 hMTE=5e1 bno=53
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=e5000 hMTE=5e1 bno=54
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=fb000 hMTE=5e1 bno=6a
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=2a000 hMTE=5e1 bno=1b
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=2c000 hMTE=5e1 bno=1d
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=27000 hMTE=5e1 bno=18
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=4d000 hMTE=5e1 bno=3e
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=55000 hMTE=5e1 bno=46
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=e0000 hMTE=5e1 bno=4f
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=10b000 hMTE=5e1 bno=7a
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=10c000 hMTE=5e1 bno=7b
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=f7000 hMTE=5e1 bno=66
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=1f000 hMTE=5e1 bno=10
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=20000 hMTE=5e1 bno=11
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=ef000 hMTE=5e1 bno=5e
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=25000 hMTE=5e1 bno=16
```

```
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrDLM entry - slot 36 ptda ab99a000
ldrDLM name  - slot 36 name H:\FAXWORKS\Fax.adp
lpi Processing imports slot=0036, module=H:\FAXWORKS\FAX.ADP
ldr walking tree hMTE=0618, name=H:\FAXWORKS\FAX.ADP
ldr walking tree going down
ldr walking tree hMTE=029a, name=H:\OS2\DLL\PMWIN.DLL
ldr walking tree going down
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0293, name=H:\OS2\DLL\PMGPI.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=029b, name=H:\OS2\DLL\MOUCALLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=01e5, name=H:\OS2\DLL\VIOCALLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0262, name=H:\OS2\DLL\NLS.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=029c, name=H:\OS2\DLL\PMSHAPI.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going down
ldr walking tree hMTE=0111, name=H:\OS2\DLL\SESMGR.DLL
ldr walking tree going up
ldr walking tree hMTE=0281, name=H:\OS2\DLL\PMMERGE.DLL
ldr walking tree going up
ldr walking tree hMTE=029a, name=H:\OS2\DLL\PMWIN.DLL
ldr walking tree going up
ldr walking tree hMTE=0618, name=H:\FAXWORKS\FAX.ADP
ldr walking tree going down
ldr walking tree hMTE=0104, name=H:\OS2\DLL\MSG.DLL
ldr walking tree going up
ldr walking tree hMTE=0618, name=H:\FAXWORKS\FAX.ADP
ldr walking tree going up
lrm, Recording init hMTE=0618, flags1=6090b1c6, name=H:\FAXWORKS\FAX.ADP
lrm, Skipping init hMTE=0104, flags1=2098b388, name=H:\OS2\DLL\MSG.DLL
lrm, Skipping init hMTE=029a, flags1=2098b388, name=H:\OS2\DLL\PMWIN.DLL
lrm, Skipping init hMTE=029c, flags1=2098b388, name=H:\OS2\DLL\PMSHAPI.DLL
lrm, Skipping init hMTE=0262, flags1=2098b388, name=H:\OS2\DLL\NLS.DLL
tk SD has-init slot=36
tk SD pre-inc slot=36 cnest=1
ldrDLM free  - slot 36
ldrDLM exit  - slot 36
ldrGP cr2=178de000 hMTE=618 bno=f
      name = H:\FAXWORKS\FAX.ADP
ldrGP cr2=178df000 hMTE=618 bno=10
      name = H:\FAXWORKS\FAX.ADP
ldrGP cr2=10e72000 hMTE=618 bno=1c
      name = H:\FAXWORKS\FAX.ADP
ldrGP cr2=178e6000 hMTE=618 bno=17
```

```
      name = H:\FAXWORKS\FAX.ADP
ldrGP cr2=178e0000 hMTE=618 bno=11
      name = H:\FAXWORKS\FAX.ADP
ldrGP cr2=10e73000 hMTE=618 bno=1d
      name = H:\FAXWORKS\FAX.ADP
ldrGP cr2=10e74000 hMTE=618 bno=1e
      name = H:\FAXWORKS\FAX.ADP
ldrGP cr2=178e5000 hMTE=618 bno=16
      name = H:\FAXWORKS\FAX.ADP
tk LIn slot=36 cnest=1
ldrGP cr2=178d1000 hMTE=618 bno=2
      name = H:\FAXWORKS\FAX.ADP
ldrGP cr2=13e51000 hMTE=281 bno=118
      name = H:\OS2\DLL\PMMERGE.DLL
ldrGP cr2=178db000 hMTE=618 bno=c
      name = H:\FAXWORKS\FAX.ADP
ldrGP cr2=10e70000 hMTE=618 bno=1a
      name = H:\FAXWORKS\FAX.ADP
ldrGP cr2=114000 hMTE=5e1 bno=83
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=26000 hMTE=5e1 bno=17
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=14000 hMTE=5e1 bno=5
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=13f7e000 hMTE=279 bno=d0
      name = H:\OS2\DLL\PMWP.DLL
ldrDLM entry - slot 36 ptda ab99a000
ldrDLM name  - slot 36 name COMETDLL
ldrDLM free  - slot 36
ldrDLM exit  - slot 36
ldrGP cr2=178c2000 hMTE=5e1 bno=91
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c0000 hMTE=5e1 bno=86
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=11000 hMTE=5e1 bno=2
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=3b000 hMTE=5e1 bno=2c
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c6000 hMTE=60b bno=7
      name = H:\FAXWORKS\FX044.LOL
ldrGP cr2=178c7000 hMTE=60b bno=8
      name = H:\FAXWORKS\FX044.LOL
ldrGP cr2=178c0000 hMTE=60b bno=1
      name = H:\FAXWORKS\FX044.LOL
ldrGP cr2=178c0000 hMTE=60b bno=1
      name = H:\FAXWORKS\FX044.LOL
ldrGP cr2=178c0000 hMTE=60b bno=1
      name = H:\FAXWORKS\FX044.LOL
ldrGP cr2=178c0000 hMTE=5e1 bno=86
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c1000 hMTE=5e1 bno=87
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c1000 hMTE=5e1 bno=87
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c1000 hMTE=5e1 bno=87
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=f6000 hMTE=5e1 bno=65
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c1000 hMTE=5e1 bno=87
```

```
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c1000 hMTE=5e1 bno=87
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c1000 hMTE=5e1 bno=87
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c2000 hMTE=5e1 bno=88
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c6000 hMTE=60b bno=7
      name = H:\FAXWORKS\FX044.LOL
ldrGP cr2=178c7000 hMTE=60b bno=8
      name = H:\FAXWORKS\FX044.LOL
ldrGP cr2=13c80000 hMTE=419 bno=34
      name = H:\OS2\DLL\HELPMGR.DLL
ldrDLM entry - slot 36 ptda ab99a000
ldrDLM name  - slot 36 name HPMGRMRI
lpi Processing imports slot=0036, module=H:\OS2\DLL\HPMGRMRI.DLL
ldr walking tree hMTE=062a, name=H:\OS2\DLL\HPMGRMRI.DLL
ldr walking tree going up
lrm, Skipping init hMTE=062a, flags1=2098b18a, name=H:\OS2\DLL\HPMGRMRI.DLL
ldrDLM free  - slot 36
ldrDLM exit  - slot 36
ldrGP cr2=178c7000 hMTE=62a bno=8
      name = H:\OS2\DLL\HPMGRMRI.DLL
ldrGP cr2=10000 hMTE=5e1 bno=1
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=113000 hMTE=5e1 bno=82
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=21000 hMTE=5e1 bno=12
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrDLM entry - slot 36 ptda ab99a000
ldrDLM name  - slot 36 name H:\OS2\DLL\HELV.FON
lpi Processing imports slot=0036, module=H:\OS2\DLL\HELV.FON
ldr walking tree hMTE=035e, name=H:\OS2\DLL\HELV.FON
ldr walking tree going up
lrm, Skipping init hMTE=035e, flags1=2098b3c8, name=H:\OS2\DLL\HELV.FON
ldrDLM free  - slot 36
ldrDLM exit  - slot 36
ldrGP cr2=28000 hMTE=5e1 bno=19
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c2000 hMTE=60b bno=3
      name = H:\FAXWORKS\FX044.LOL
ldrGP cr2=42000 hMTE=5e1 bno=33
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=44000 hMTE=5e1 bno=35
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=41000 hMTE=5e1 bno=32
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c0000 hMTE=60b bno=1
      name = H:\FAXWORKS\FX044.LOL
ldrGP cr2=178c6000 hMTE=60b bno=7
      name = H:\FAXWORKS\FX044.LOL
ldrGP cr2=178c7000 hMTE=60b bno=8
      name = H:\FAXWORKS\FX044.LOL
ldrGP cr2=43000 hMTE=5e1 bno=34
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=40000 hMTE=5e1 bno=31
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=29000 hMTE=5e1 bno=1a
```

```
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=53000 hMTE=5e1 bno=44
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=3d000 hMTE=5e1 bno=2e
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=112000 hMTE=5e1 bno=81
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=12000 hMTE=5e1 bno=3
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c2000 hMTE=5e1 bno=91
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c2000 hMTE=5e1 bno=91
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c0000 hMTE=5e1 bno=8f
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c0000 hMTE=5e1 bno=8f
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c0000 hMTE=5e1 bno=8f
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c0000 hMTE=5e1 bno=8f
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c0000 hMTE=5e1 bno=8f
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c0000 hMTE=5e1 bno=8f
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c0000 hMTE=5e1 bno=8f
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c1000 hMTE=5e1 bno=90
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c2000 hMTE=5e1 bno=91
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c1000 hMTE=5e1 bno=90
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c1000 hMTE=5e1 bno=90
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c1000 hMTE=5e1 bno=90
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c1000 hMTE=5e1 bno=90
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c1000 hMTE=5e1 bno=90
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c1000 hMTE=5e1 bno=90
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c1000 hMTE=5e1 bno=90
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c1000 hMTE=5e1 bno=90
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c2000 hMTE=5e1 bno=91
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c2000 hMTE=5e1 bno=91
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178c6000 hMTE=60b bno=7
      name = H:\FAXWORKS\FX044.LOL
ldrGP cr2=178c6000 hMTE=60b bno=7
      name = H:\FAXWORKS\FX044.LOL
ldrGP cr2=22000 hMTE=5e1 bno=13
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=24000 hMTE=5e1 bno=15
```

```
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=37000 hMTE=5e1 bno=28
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=23000 hMTE=5e1 bno=14
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=13e52000 hMTE=281 bno=119
      name = H:\OS2\DLL\PMMERGE.DLL
ldrGP cr2=13e53000 hMTE=281 bno=11a
      name = H:\OS2\DLL\PMMERGE.DLL
ldrGP cr2=13e54000 hMTE=281 bno=11b
      name = H:\OS2\DLL\PMMERGE.DLL
```

The following shows the loader sequence when FAXWORKS.EXE is terminated:

```
ldrGP cr2=178c6000 hMTE=60b bno=7
      name = H:\FAXWORKS\FX044.LOL
ldrGP cr2=178c7000 hMTE=60b bno=8
      name = H:\FAXWORKS\FX044.LOL
ldrGP cr2=178c6000 hMTE=60b bno=7
      name = H:\FAXWORKS\FX044.LOL
ldrGP cr2=178c6000 hMTE=60b bno=7
      name = H:\FAXWORKS\FX044.LOL
ldrGP cr2=50000 hMTE=5e1 bno=41
      name = H:\FAXWORKS\FAXWORKS.EXE
ldrGP cr2=178e4000 hMTE=618 bno=15
      name = H:\FAXWORKS\FAX.ADP
```

1.4.5 DosDebug Logging Facility

The kernel worker routines for the *DosDebug* API implement a number of logging functions for use in debugging errors in *DosDebug* itself. These are activated by setting bits in the double-word at symbol: *_DBGbugbug*.

The following flags bits are defined:

0x01000000
Display input to DosDebug

0x02000000
Display output from DosDebug

0x00000010
Display exceptions in DosDebug processing

0x10000000
Display execution flow in debugger processing

0x20000000
Display execution flow in debugger processing

0x40000000
Display execution flow in watchpoint and debug register processing

1.4.6 DosPTrace Logging Facility

The kernel worker routines for the *DosPTrace* API implement a number of logging functions for use in debugging errors in *DosPTrace* itself. These are activated by setting bits in the doubleword at symbol: *_PTbugbug*.

Note: *DosPTrace* internally thunks to *DosDebug* therefore DosDebug Logging Facility may be a useful diagnostic aid with *DosPTrace*.

The following flags bits are defined:

0x01000000
Display output buffer setup passed to user

0x02000000
Display input buffer setup passed from user

0x04000000
Display conversion routine flow

0x08000000
Display alias conversion routine flow

0x10000000
Display input and output return codes only

0x20000000
Display processing of notifications from DosDebug

0x00000001
Display floating point information

1.5 Kernel Debugger Breakpoints

The breakpoint command set of the Kernel Debugger provides a mechanism for intercepting the execution of code through a particular path. For debugging application programs, breakpoints are generally required within the application itself or on call to or return from one or more system APIs.

Each system API results either in a call to a system DLL or to the Kernel through a Call Gate. The name of a system interface that is called when an application uses an API is either identical to the API name or may be determined from one of the following conventions:

DosI*name* Kernel Call Gate name corresponding to API Dos*name*.

Dos32*name* DOSCALL1 32-bit entry point corresponding to API Dos*name*.

Dos16*name* DOSCALL1 16-bit entry point corresponding to API Dos**name**.

Other system DLLs such as PMWIN.DLL, PMMERGE.DLL, etc. adopt similar conventions, for example, API *WinCreateWindow* calls *Win32CreateWindow* in PMMERGE.DLL.

In nearly all cases the system entry points have corresponding system trace points with the entry point name prefixed with either pre or post. So the *System Tracepoints Reference (Volume 4)* provides a comprehensive source for for deriving API related breakpoints.

Physical Device Driver helper routines pass through a common router, then to specific worker routines. Worker entry point names generally adhere to the following convention:

DosHlp_*name* worker routine dh_*name*.

Virtual Device Driver helper routines have entry points in the kernel with identical names (folded to uppercase) to the helper name.

File system driver and mini-file system driver helper routines have entry points in the kernel with identical names to the helper name.

In addition to API and driver helper related breakpoints, the following system labels may also prove useful when intercepting errors or program initiation:

_tkSchedNext

This routine is called when a new thread is selected for scheduling. The out-going thread slot number is recorded in variable *Tasknumber*.

_tkSchedNext exits from one of two points:

SchedNextRet A new thread slot is selected.

SchedNextRet2 The same thread slot is selected.

These labels maybe used to obtain a trace of dispatching activity. This is particularly useful when trying to establish the scope of hang conditions.

The following example illustrates how to obtain a trace of dispatched tasks using this breakpoint.

```
##bp _tkschednext,".p #;g"
##g
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
*0033# 0019 0000 0019 0001 blk 081e 7b98c000 7bb4b288 7bb2d394 1bf8 10 wkstahlp
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
*0034# 0018 0000 0018 0002 run 021f 7b98e000 7bb4aa5c 7bb2d548 1ea8 10 wksta
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
*0008# 0006 0001 0006 0001 blk 0500 7b936000 7bb460d0 7bb28a58 1eb8 01 pmshell
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
*0034# 0018 0000 0018 0002 blk 021f 7b98e000 7bb4aa5c 7bb2d548 1f00 10 wksta
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
*0038# 0018 0000 0018 0003 blk 0200 7b996000 7bb4aa5c 7bb2dc18 1eb8 10 wksta
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
*0008# 0006 0001 0006 0001 blk 0500 7b936000 7bb460d0 7bb28a58 1eb8 01 pmshell
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
*000b# 0004 0000 0004 0001 blk 080b 7b93c000 7bb45078 7bb28f74 1cf0 00 landll
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
*0008# 0006 0001 0006 0001 blk 0500 7b936000 7bb460d0 7bb28a58 1eb8 01 pmshell
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
*0008# 0006 0001 0006 0001 blk 0500 7b936000 7bb460d0 7bb28a58 1eb8 01 pmshell
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
*0008# 0006 0001 0006 0001 blk 0500 7b936000 7bb460d0 7bb28a58 1eb8 01 pmshell
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
*0008# 0006 0001 0006 0001 blk 0500 7b936000 7bb460d0 7bb28a58 1eb8 01 pmshell
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
*0008# 0006 0001 0006 0001 blk 0500 7b936000 7bb460d0 7bb28a58 1eb8 01 pmshell
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
*0008# 0006 0001 0006 0001 blk 0500 7b936000 7bb460d0 7bb28a58 1eb8 01 pmshell
```

Note: The status is *blocked* since *_tkSchedNext* has been called because the current thread is giving up its time-slice.

DosLibIDisp

This API is called to initiate DLL initialization whenever a new module is loaded into memory. Since this is called for every .EXE at load time, in the context of the new process and thread, it provides an excellent breakpoint for intercepting the loading of a new module in a new process.

When DosLibIDisp receives control MTE and SMTE have been created and the program module has been loaded. From the SMTE we can determine the entry point of the new module and thus set a breakpoint on this address.

The following example illustrates how to set a breakpoint on entry to a new module.

```
>> Add breakpoint at DosLibIDisp, then start CMD.EXE

##bp doslibDisp
##g
eax=00000000 ebx=000029f4 ecx=00000010 edx=00000014 esi=00000bc8 edi=00000c0a
eip=00000294 esp=0000773c ebp=00007752 iopl=2 -- -- -- nv up ei pl nz na po nc
cs=ffd7 ss=001f ds=ffa7 es=ffaf fs=150b gs=0000  cr2=1fc70490  cr3=001d0000
doscall1:CODE16_GROUP:DOSLIBIDISP:
ffd7:00000294 b80100         mov     ax,0001          ;br0

##.p#
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
*0044# 002c 0006 002c 0001 run 0400 7b9ae000 7bb4fc14 7bb2f088 1f48 19 cmd


>> The hmte for the current process is found in the PTDA at
>> ptda_module

##dw ptda_module l1
0030:0000ffaa  03a1

##.lmo 3a1
hmte=03a1 pmte=%fe97ebe4 mflags=84903152 c:\os2\cmd.exe
obj   vsize    vbase    flags   ipagemap cpagemap hob  sel
0001 0000c6a8 00010000 80001025 00000001 0000000d 03a0 000f r-x shr alias
0002 00007efa 00020000 80001025 0000000e 00000008 03a2 0017 r-x shr alias
0003 00009730 00030000 80001043 00000016 00000002 0000 001f rw- prel alias



>> Now dump the MTE and SMTE, whose address is at MTE+0x4

##dd %fe97ebe4 l8
%fe97ebe4  03a10002 fd4341d0 fe97ec1c fe9a143c
%fe97ebf4  84903152 00000007 00060050 fe908e74

##dd %fd4341d0
%fd4341d0  00000017 00000002 000044fa 00000003
%fd4341e0  00007790 00000009 000005d9 fd434261
%fd4341f0  00000003 fd4342a9 00000a00 00000000
%fd434200  00000000 fd434361 fd434368 fd434369
%fd434210  fd4343c9 fd4348f1 fd434924 00000a00
%fd434220  00000000 00000000 00000003 00000000
%fd434230  00000000 00001fa0 fd434252 00000000
%fd434240  00000000 00003f40 00000000 0000000e
```

```
>> SMTE+0x4 is the entry point object number
>> SMTE+0x8 is the entry point offset offset
>> For CMD.EXE this is 2:44fa
>> Since object 2 starts at %20000, we can define a breakpoint on
>> entry to CMD.EXE at %20000+44fa

##bp %00020000 +44fa

##bl
0 e  ffd7:00000294 [DOSLIBIDISP]
1 e  %000244fa [__astart]

>> Disable BP 0 since DosLibIDisp is called for every DLL that will be
>> initialised in the new process.

##bd 0
##g
eax=00000027 ebx=00000491 ecx=00009730 edx=0000f834 esi=00001fa0 edi=000003a1
eip=000044fa esp=00007790 ebp=00000000 iopl=2 -- -- -- nv up ei pl nz na po nc
cs=0017 ss=001f ds=001f es=0000 fs=150b gs=0000  cr2=00063ffe  cr3=001d0000
cmd:_TEXT3:__astart:
0017:000044fa fc             cld                      ;br2
```

VMLockMem

This breakpoint is on entry to the memory locking subroutine of Virtual Memory Management. It may be used in conjunction with the VM Lock Trace.

_XCPTBuildR3DispatcherStack

This routine is called whenever a process fatal exception is generated, regardless of whether exception handlers are registered. It therefore makes a stronger method than *VSF ** for intercepting fatal user exceptions.

Exception management and how to intercept exceptions is discussed in more detail in the Trap and Exception Processing section.

_xcptR3ExceptionDispatcher

Whenever an exception (that is not fatal to the system), the Ring 3 Exception Dispatcher is called to dispatch registered exceptions. It does this by locating exception registration records from the TIB at +0x0. :1i.Thread Information Block

On entry to the Ring 3 Exception Dispatcher, *ESP+0x4* and *EXP+0x8* point to the exception report record and exception context record, respectively.

The exception report record contains the exception number, and exception address.

The exception context record contains all register values at the time of exception.

The layout for both these records is given in the BSEXCPT.H header file of the OS/2 Programmer's Toolkit.

Most exceptions are generated from a hardware detected exception such as a trap. These are readily intercepted by using the Kernel Debugger VSF command. Exceptions may also be generated by the DosRaiseException API. Whatever the source all exceptions will eventually result in a call to *_xcptrR3ExceptionDispatcher*. This makes this label an excellent breakpoint

for intercepting and filtering any exception that will drive a user's exception handler.

The following example illustrates the use of this breakpoint, where the system generates a *C0000005* exception following a Trap E in an application program.

```
>> Beak on entry to the Ring 3 Exception Handler Dispatcher
##bp _xcptr3exceptiondispatcher

>> Intercept all fatal exceptions
##vsf *
##g
Symbols linked (trape)
Trap 14 (0EH) - Page Fault 0004, Not Present, Read Access, User Mode
eax=00000000 ebx=00000000 ecx=0002059c edx=000a0000 esi=00000000 edi=00000000
eip=0001011c esp=00022e6c ebp=00022e74 iopl=2 rf -- -- nv up ei pl nz ac pe nc
cs=005b ss=0053 ds=0053 es=0053 fs=150b gs=0000  cr2=00000000  cr3=001d0000
005b:0001011c 8b00          mov     eax,dword ptr [eax]    ds:00000000=invalid

>> A fatal exception has been intercepted at %1011c
>> Now GT and see the exception dispatcher called.

##gt
eax=00022d18 ebx=00000000 ecx=0002059c edx=000a0000 esi=00000000 edi=00000000
eip=1ff9c8d8 esp=00022bf0 ebp=00022d04 iopl=2 -- -- -- nv up ei pl zr na pe nc
cs=005b ss=0053 ds=0053 es=0053 fs=150b gs=0000  cr2=00000000  cr3=001d0000
doscall1:FLAT32:_xcptR3ExceptionDispatcher:
005b:1ff9c8d8 55            push    ebp             ;br0


>> %ESP+4 points to the exception report record
>> %ESP+8 points to the exception context record

##dd %esp
%00022bf0  1ff9c7e9 00022d18 00022d3c 00000000
%00022c00  00000000 2c1a0002 154b0000 00100000
%00022c10  00010002 00000000 032b0000 ffa72212
%00022c20  0058ffaf 2c520066 154b0000 033e0002
%00022c30  52110000 ff9f3130 00000000 00172c52
%00022c40  e91f0000 e9270116 ffa70066 3029ffa7
%00022c50  0008ffa7 e9170000 00570000 00000019
%00022c60  00008000 00000000 00f80000 80000000


>> The exception report record contains the exception code at
>> offset +0x0 (in this case C0000005).
>> At offset +0xc is the address at which the exception occurred.
>> This agrees with the address seen after VSF intercepted the fatal
>> exception.

##dd %00022d18
%00022d18  c0000005 00000000 00000000 0001011c
%00022d28  00000002 00000001 00000000 7bb4fc94
%00022d38  ffdf9264 00000007 0000699c fff5416b
%00022d48  00000433 ffdf9264 7b9afe0c ffdf6378
%00022d58  00000433 000069bc fff54ef2 ffdf9264
%00022d68  ff1f5a50 fe86106c 00000000 fed022d0
%00022d78  00000000 00006a04 fff6d8d9 00000053
%00022d88  00000000 7b9afe3c ff1f5a50 7cf8014c
```

```
>> The context record contains the registers at time of exception.
>> Note the cs:eip at +0xa0 and +0x9c. Also the ss:esp at +0xbc abd
>> +0xb8 and ebp at +0x98.

##dd %00022d3c
%00022d3c  00000007 0000699c fff5416b 00000433
%00022d4c  ffdf9264 7b9afe0c ffdf6378 00000433
%00022d5c  000069bc fff54ef2 ffdf9264 ff1f5a50
%00022d6c  fe86106c 00000000 fed022d0 00000000
%00022d7c  00006a04 fff6d8d9 00000053 00000000
%00022d8c  7b9afe3c ff1f5a50 7cf8014c 7cf80088
%00022d9c  00000001 7cf80088 00000001 7bb2fdb2
%00022dac  00000000 0000150b 00000053 00000053
##d
%00022dbc  00000000 00000000 00000000 00000000
%00022dcc  0002059c 000a0000 00022e74 0001011c
%00022ddc  0000005b 00012216 00022e6c 00000053
%00022dec  00060210 00000000 00000000 000205fc
%00022dfc  000205fc 00020a40 00000000 00000000
%00022e0c  00000000 00000000 00000000 000a0000
%00022e1c  00002000 00000000 00090000 00022e50
%00022e2c  00011fa8 000205fc 00090000 00000000
```

Dos32Exit and DosR3ExitAddr

Both these labels provide good breakpoints to catch an application terminating normally.

Dos32Exit is the entry point for the DosExit API.

DosR3ExitAddr is the entry point in DOSCALL1.DLL, called when an application issues the return statement to return to the system.

Win32SetErrorInfo

This API is called by PM whenever it needs to record a PM error. When this is used as a breakpoint, the doubleword at *%esp+0x4* contains the PM error code about to be recorded.

NWDHandler

This symbol is the entry point to the Trap 2 interrupt handler. The IDT entry for Trap 2 contains a Task Gate that points to NWDHandler. When NWDHandler receives control the Task Register will contain the selector for the current TSS. The link field of the current TSS will contain the previous value of the TR, where the processor saved the current registers when the interrupt occurred.

Frequently NMI interrupts are associated with disabled code and obscure hardware or software problems. If can be useful on these occasions to set up a KDB.INI file with the following commands to display information when the trap 2 occurs. This is particularly advantageous when dealing with NMI interrupts caused by the NMI Watch Dog timer firing.

```
bp nwdhandler,"? 'curr tss';dt tr:0;? 'prev tss';dt #(wo(tr:0)):0"
```

Note: When the first NMI occurs, the following would be displayed:

```
curr tss

eax=00000000 ebx=00000000 ecx=00000000 edx=00000000 esi=00000000 edi=00000000
eip=fff4074c esp=00000400 ebp=00000000 iopl=0 -- -- -- nv up di pl nz na po nc
cs=0170 ss=1ea0 ds=0168 es=0168 fs=0000 gs=0000  cr3=001dd000
ss0=0000  esp0=00000000  ss1=0000  esp1=00000000  ss2=0000  esp2=00000000
ldtr=0000  link=0010  tflags=0000  i/o map=ffff
ports trapped: 0-ffff
prev tss

eax=000002ff ebx=139b0000 ecx=00000400 edx=00009ae8 esi=139b993c edi=139d0400
eip=1b7228fe esp=0006eeaa ebp=0006eee0 iopl=2 -- -- -- nv up ei pl nz na po nc
cs=005a ss=004a ds=0053 es=0053 fs=150b gs=0000  cr3=001dd000
ss0=0030  esp0=00006d80  ss1=0000  esp1=00000000  ss2=0036  esp2=0000f000
ldtr=0028  link=0000  tflags=0000  i/o map=dfff
ports trapped: 0-ffff
##
```

The register values when the NMI occurred are displayed under the label *prev tss*.

After NWDHandler has processed the NMI, the NMI TSS is edited and the entry point on subsequent NMI is approximately NWDHandler+25. This may be used as an indication that an NMI has previously occurred.

1.6 Trap and Exception Processing

The fine detail of exception management by OS/2 is complex. However the principles are easy to grasp. This section gives an overview of OS/2 exception management sufficient to provide the reader with a technique for intercepting exceptions in user code under the Kernel Debugger.

Exception Definition

Exceptions may be summarized as follows:

- Exceptions refer either to:

  Hardware Traps and Faults - INTEL defined.

  Software generated exceptions - OS/2 and user defined.

- Each hardware exception has an associated vector, which the processor uses to index the IDT to give control to the appropriate system exception handler.
- OS/2 converts traps and faults to software exceptions. For example, Traps 0xd and 0xe are converted to exception 0xc0000005.
- Software exceptions are generated from three sources:

  1. Converted hardware Traps and Faults.
  2. Software signals.
  3. Software exceptions from DosRaiseException

- Exceptions occur for both normal and abnormal reasons. In the normal case additional processing is required to be executed in a manner transparent to the main line code. Examples of this are:

  Page Fault exceptions.

Trap 1 and 3 for system trace

387 co-processor emulation

VDM privileged instruction emulation

In the abnormal case, an error condition has been detected. If the error cannot be corrected then either a process or the system dies depending on whether the error can be isolated to a particular process. Usually traps and faults in ring 0 code result in system termination. Bad parameters passed in system APIs may cause the kernel to trap. The system recovers by directing an exception 0xc0000005 to a process. Unless the process can handle this exception, it dies.

- Full details of OS/2 defined exceptions are given in OS/2 System Exception Codes.

Exception Logic

The essential logic for exception handling is as follows:

- If the processor generates a hardware exception then control is given to the first level exception handler pointed to by the IDT descriptor that corresponds to the hardware exception vector.
- If the Kernel Debugger Vector Commands have been specified without the fatal flag then first level exception handlers have been replaced by the Kernel Debugger routines. These may give control to the debugging console or enter the normal system handlers if interception criteria are not satisfied.
- The non-debugger first level routines perform any specific processing for the current exception, for example processing single step and breakpoint traps.
- If full recovery is possible then the first level routines exit with an IRET instruction.
- In most cases control passes from the first level trap handlers to *TrapCommonFaultEntry*. This performs common processing for all hardware exceptions. If recovery is possible, for example by satisfying a page fault or making a segment present, then this is done and control returned to the interrupted code.

  If recovery is not directly possible or further special processing is required then control passes to one of the following second level exception handlers:

  V8086 emulation for instruction emulation.

  VDM exception handler to reflect non-fatal exceptions back to the VDM using its IDT.

  Process fatal fault handler (*_TRAPProcessFatalFault*) for non-kernel mode code (InDos=0).

  Kernel fault handler for kernel code (InDos=1)

  Special handlers for co-processor handling, NMIs etc..
- The kernel fault handler checks for the presence of a local fault handler by inspecting *TSDpfnFault*. If this is non-zero then passes control to the local fault handler, otherwise it passes to the system fatal fault handler (*SystemFatalFault*).
- Ths system fatal fault handler will enter the Kernel Debugger (if in a non-RETAIL kernel), otherwise it will call and device drivers that have registered for notification of fatal system faults, then exit to the panic routine

with a formatted message - usually the IPE trap screen. Once in panic the system will not dispatch any more threads. If TRAPDUMP or REIPL are specified then these are acted on otherwise the system waits to be re-booted.

- The process fatal fault handler will check for fatal fault interception by the Kernel Debugger and enter the kernel debugger if interception criteria are satisfied. Otherwise *DelayHardErr* is called to build the trap screen and wake the hard error process. Control then passes to the *_XCPTBuildR3DispatcherStack*.
- *_XCPTBuildR3DispatcherStack* is responsible for massaging the users stack so that when the kernel exits, control returns to the exception dispatcher (*_xcptr3ExceptionDispatcher* in DOSCALL1.DLL). If no exception registration records exist for the current thread then the thread enters termination and the exception dispatcher is not called.
- The exception dispatcher runs the chain of exception registration records, anchored from the TIB of the current thread. Each registered user exception handler is called in turn (via an intermediate routine, *_xcptExecuteUserExceptionHandler*). The return code passed back by the exception handle is examined. If it specifies *XCPT_CONTINUE_EXECUTION* then control returns to the kernel via *Dos32ExceptionCallBack*, whereupon the thread′s stack is prepared for returning to the interrupted program. If *XCPT_CONTINUE_SEARCH* is specified then the next exception handler in the chain is dispatched. When the last exception handler has been dispatched (and all have returned *XCPT_CONTINUE_SEARCH*) then control passes to the kernel via *Dos32ExceptionCallBack* and the thread is terminated.
- Local Fault Handlers are exception handlers registered by kernel routines. Typically one is registered on entry to the kernel by an API call, and de-registered on exit. If a local fault handler cannot resolve the fault then it will call panic if a serious system fault has occurred, or *_XCPTBuildR3DispatcherStack* if user code is at fault. For example, when a bad parameter supplied to an API by an application program causes the kernel to Trap.
- Unlike user exception handlers, local exception handlers are not allowed to recurse and at most, only one can be registered. When the system calls a local exception handler, *TSDpfnFault* is zeroed, thereby de-registering it. Local exception handlers are always deregistered on exiting the kernel.
- The *DosRaiseException* API is called to create a user exception. This passes control to *_XCPTBuildR3ExceptionDispatcher* and normal user exception processing follows.

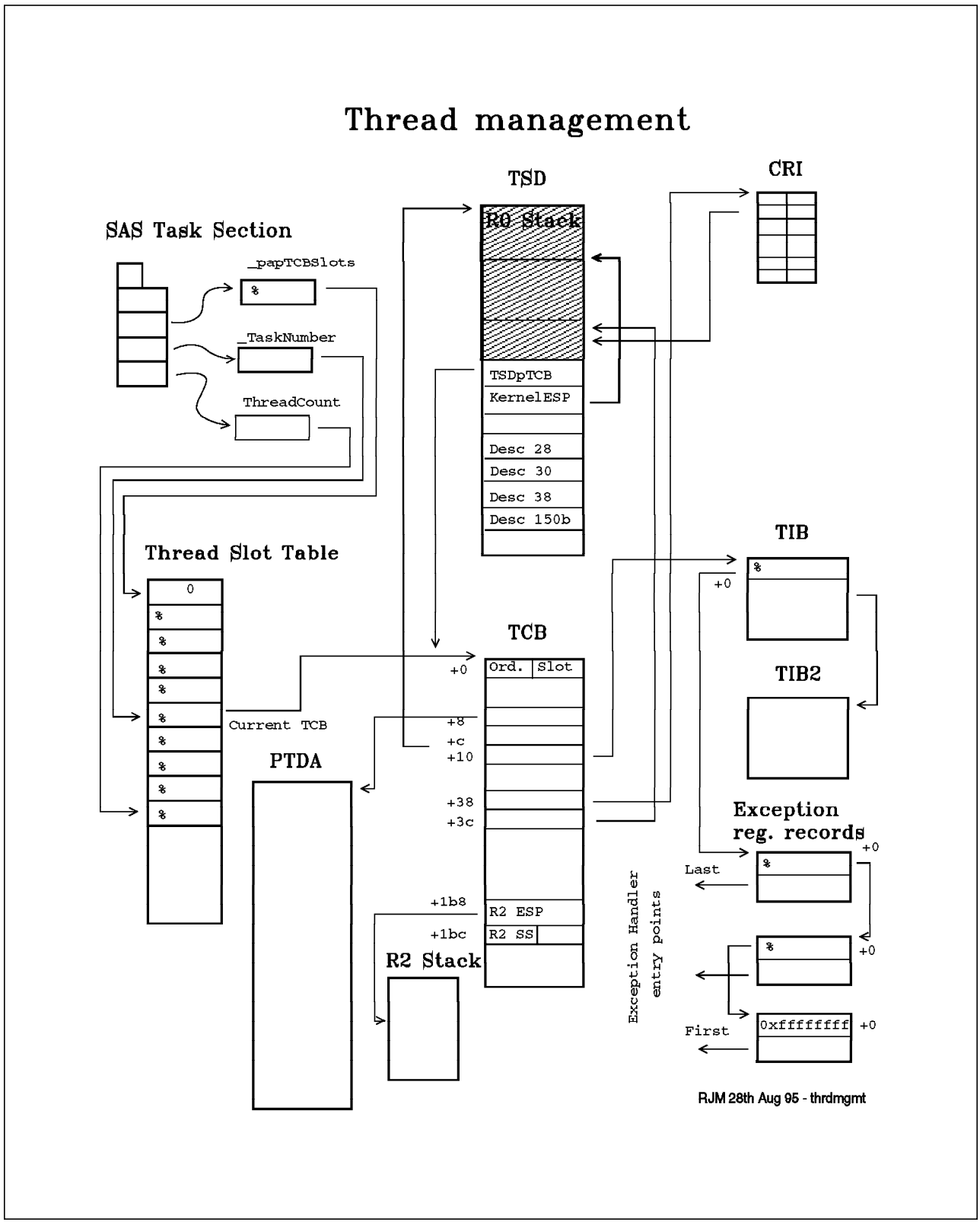
These details are summarized in the following diagrams:

1.6.1, “Exception Registration Records” on page 47.

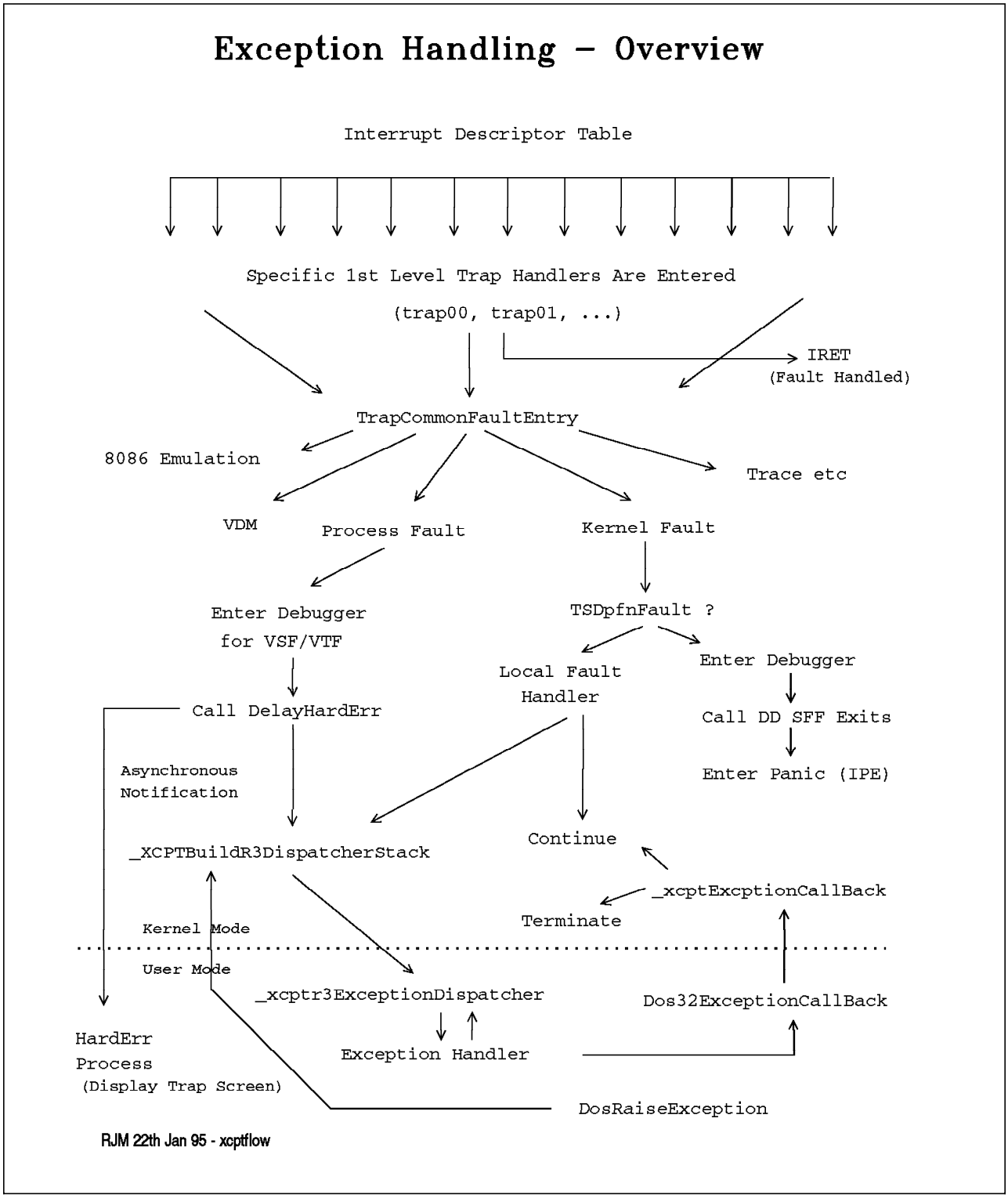
1.6.2, “OS/2 Exception Exception Management - Overview” on page 48.

1.6.3, “Exception Handler Stack Frames” on page 49.

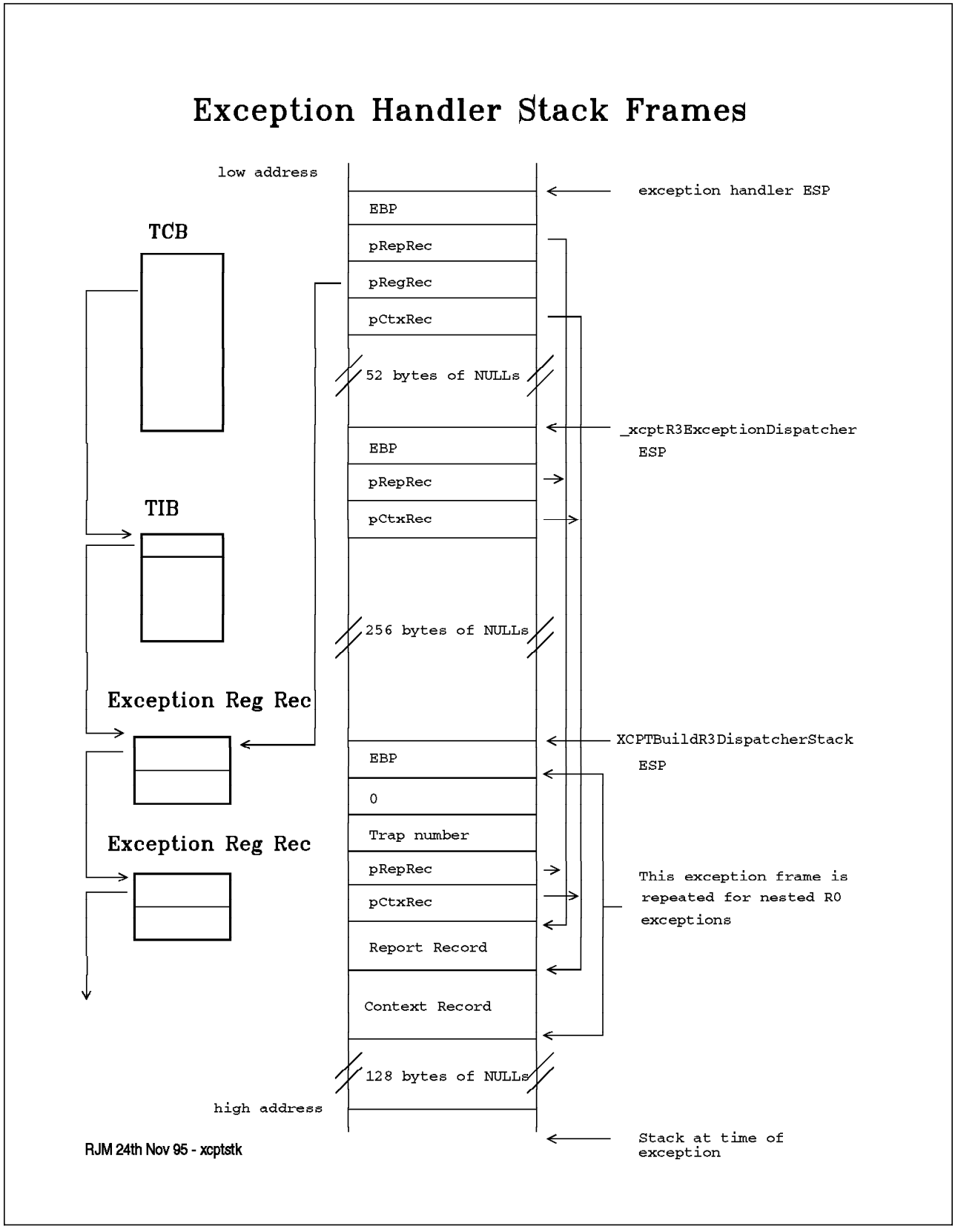
1.6.1 Exception Registration Records



1.6.2 OS/2 Exception Exception Management - Overview



1.6.3 Exception Handler Stack Frames



1.6.4 Intercepting Exceptions and Traps

The following list provides guidelines for intercepting Traps and exceptions under the Kernel Debugger for various circumstances:

Fatal exceptions occurring in application ring 2/3 code.
BP _XCPTBuildR3DispatcherStack will trap every software and hardware exception. The breakpoint is in the kernel, so use **.**R to display the registers at the time of the exception. This break-point works regardless of whether exception handlers are registered.

Note: If the exception is generated through use of an API (bad parameter or *DosRaiseException*) then the CS:EIP will point after the call gate instruction.

Fatal hardware traps and faults in application ring 2/3 code.
VSF * will intercept all such exceptions at the point of the exception.

Fatal hardware traps and faults in ring 0 code.
VTF * will intercept all such exceptions at the point of the exception, providing no Local Fault Handler has been registered.

All ring 0-3 traps and faults.
VT * will intercept them all.

All application ring 2/3 code traps and faults.
VS * will intercept them all.

Exceptions in application ring 2/3 code that will drive exception handlers.
BP _xcptr3ExceptionDispatcher will be intercepted if any are registered, but this will be called once to process the entire chain.

Each User Exception Handler.
BP _xcptExecuteUserExceptionHandler will be called to dispatch each exception handler. Alternatively use the registration records from the TIB to locate the entry point of a given exception handler.

Note: User exception handlers can be disabled under the Kernel Debugger by locating the TIB, then storing 0xffffffff at offset 0x0, which is the pointer to the exception registration record chain. The chain is terminated by 0xffffffff and can be re-worked manually for debugging purposes - provided that the system is not already processing an exception for this thread.

---

[2] Throughout this chapter the term **debugger**. is used loosely to mean any of the following where ambiguity is not a problem:

Debug Kernel (HSTRICT or ALLSTRICT).

The debugger component within the debug kernel.

The debugging console.

Chapter 2. Dump Formatter User Guide

The Dump Formatter is an interactive line-mode utility that supports a variety of commands for extracting and displaying information from a system dump. There are two versions of the Dump Formatter:

DF_RET.EXE The Dump Formatter for dumps from systems running either the RETAIL or HSTRICT kernels.

DF_DEB.EXE The Dump Formatter for dumps from systems running the ALLSTRICT kernel.

Note: Refer to the Chapter 1, “Kernel Debugger User Guide” on page 1 for a discussion on the different OS/2 Kernels.

Each of the two Dump Formatters is generated for each build of the OS/2 kernel. Thus the Dump Formatter is system level and FixPak level dependent, in a similar way to the debug kernels. Several base versions of the Dump Formatters are distributed with the OS2PDP package. For versions that correspond to a particular FixPak, contact your local IBM Service Representative.

The Dump Formatter has a named pipe interface that allow it to be controlled from another program. This is exploited by the PMDF program, which is also distributed with the OS2PDP package.

PMDF provides:

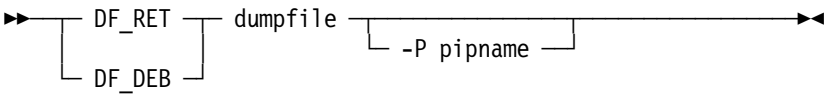
- A PM interface to the Dump Formatter
- Automatic Dump Formatter version management
- The ability to log output to a file
- Use of Drag and Drop on Dump Formatter output to the PMDF commands line
- A REXX interface that allows REXX EXECs to issue Dump Formatter commands and capture their output
- Process Dump Formatting

The command set supported by Dump Formatter is very similar to that of the Kernel Debugger. In many cases they share common commands. These are documented in Chapter 3, “Kernel Debugger and Dump Formatter Command Reference” on page 71.

2.1 Dump Formatter Installation

The Dump Formatter may be installed together with PMDF by using the installation procedure supplied with the OS2PDP package. Alternatively copy the *.EXE files to either a private directory or a directory in your current PATH. The only files the Dump Formatter accesses implicitly are symbol files, which if used, are convenient to have installed in the same directory as the .EXE program files.

The command line syntax for the Dump Formatter is as follows:



The parameters have the following meaning:

dumpfile
The file name of the (decompressed) dump to be analyzed. If a path is not prepended to the file name then the Dump Formatter assumes the current path. See 2.2, “Dump Decompression” on page 53.

-P*pipename*
The name of a named pipe through which Dump Formatter output and commands are channeled.

Note: This parameter is intended for use when DF&US.RET.EXE or DF&US.DEB.EXE is started from another program using the DosExecPgm API.

Note:

If no parameters are entered then the Dump Formatter give a syntax message. This message implies that a COM port may also be used as an interface, but this has not been implemented.

When the Dump Formatter is started it displays the build level of the system from which the dump was taken and then the build level of the formatter. If these do not match unpredictable, results may occur. However, if the levels are close then it is probably safe to use the Dump Formatter, though not guaranteed.

If the incorrect type of Dump Formatter is used, for example, RETAIL Dump Formatter with a ALLSTRICT dump, then the Dump Formatter will probably trap. If it does not, then an error message will appear.

In general the Dump Formatter traps for one of three reasons:

- The reasons stated above, where there are type and level mismatches.
- The dump file is incomplete or corrupted.
- The Dump Formatter stack overflows.

The latter problem usually occurs when the **.**P is used. This is sometimes circumventable by using the EXEDHR utility to increase the stack size of the Dump Formatter. Another approach is to use the %PS REXX to display each thread slot individually.

As part of the initialization sequence, the Dump Formatter attempts to load symbol files, from the current directory, for each module that was loaded on the dumped system.

Notes:

Windows, WINOS2 and DOS symbol files are not usable under the Dump Formatter However, the *SYMLST* REXX exec in the tools directory of the accompanying CD-ROM may be used to list a symbol file. This can sometimes be used in conjunction with the Dump

Formatter provided that at least one location of a module or its data can be determined absolutely.

Symbol files not present in the current directory may be manually loaded using the WA command. The syntax and function of this command differs subtly from the Kernel Debugger equivalent:

- Under Dump Formatter names are symbol file names unlike Kernel Debugger where they are symbol map names. This allows relative path names to be used.
- Under Dump Formatter WA reads the symbol file, whereas under Kernel Debugger it is just marked active provided it was loaded when the module was loaded.

The Dump Formatter prompts for command input with a single # sign. Unlike the Kernel Debugger this is not used to signify the processor mode or whether paging is enabled. Consequently the Dump Formatter always assumes that the current processor mode is Protect Mode with Paging Enabled. The user must therefore explicitly prefix **segment:offset** addresses in Virtual 8089 mode with an ampersand (&).

Commands may be interrupted by pressing the Esc key.

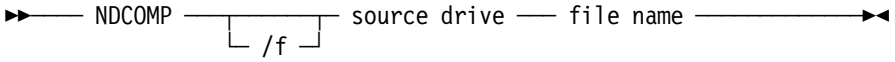
2.2 Dump Decompression

Dumps may be taken either to a dedicated FAT hard disk partition or to diskette. For details on setting up the dump partition refer to the TRAPDUMP CONFIG.SYS command description.

Dumps taken to a hard disk partition may be used directly by Dump Formatter or PMDF.

Dumps taken to diskette have their data compressed and have to be decompressed to produce a single dump file. This may be done directly from within PMDF by selecting the *New* option of the *File* pull-down menu. PMDF offers the additional facility of decompressing diskette dumps directly from diskette images created by OS2IMAGE. See 2.4.1, “PMDF File Menu” on page 55 for details. Sometimes PMDF fails to decompress a dump, in which case, the *NDCOMP* utility may be used.

NDCOMP has a better tolerance for missing dump data or corrupted dumps. The syntax for NDCOMP is as follows:



/f

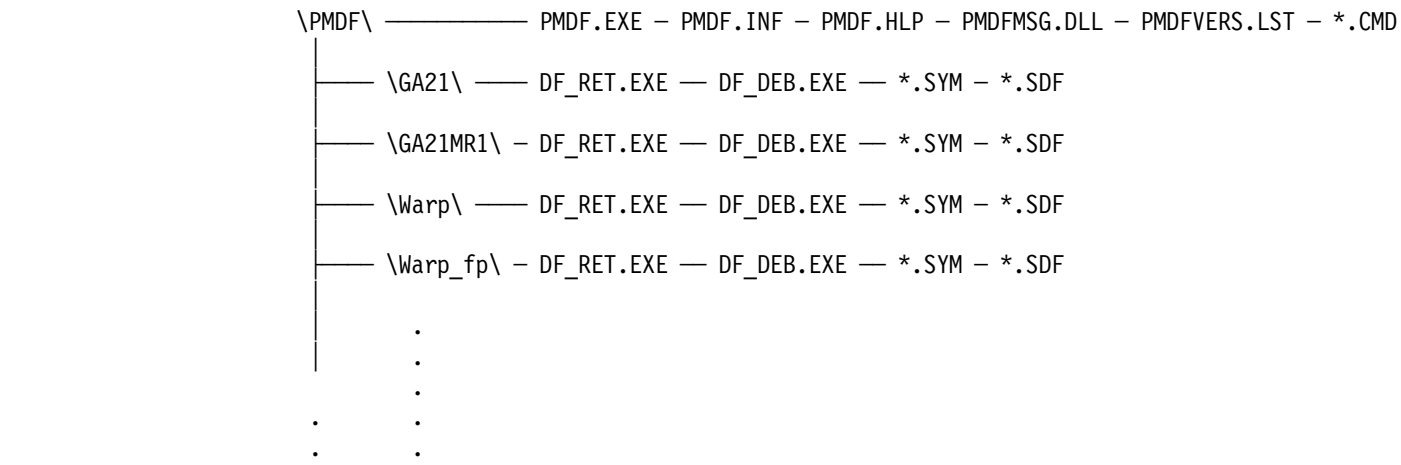
source drive
Specifies the drive where the *DUMPDATA.nnn* file will be found. This may specify either a hard disk drive or a diskette drive. The *DUMPDATA.nnn* files from a diskette dump may be copied to a hard drive root directory before using the NDCOMP utility.

file name
The target dump file name including path information.

2.3 Presentation Manager Dump Formatter (PMDF) Installation

PMDF provides a convenient front-end to the Dump Formatter. By installing PMDF in an appropriate directory structure it is able to select automatically the correct version of Dump Formatter for the dump to be analyzed.

PMDF is installed by using the installation of the *OS2PDP* package. The resulting directory structure is as follows:



Each version of OS/2 is represented by a subdirectory containing the Dump Formatters and symbol files and structure definition files for that version.

The home directory contains the PMDF executables and help files, the version control file PMDFVERS.LST, and any REXX EXECs to be installed in their default directory.

PMDFVERS.LST

More versions of the Dump Formatter may be installed by creating a new subdirectory for the new Dump Formatter and adding an entry to the PMDFVERS.LST file. Each entry of this file corresponds to an OS/2 build level or version. The format of an entry is as follows:

```
relative path:build level:descriptive text
```

Notes: The path is relative to the home directory.

The build level is the internal system build level and may be determined either by browsing the OS2KRNL load module and searching for the text
`@#IBM:n.nnn#@`
near the end of the module, or by using the BLDLEVEL utility in the OS2 directory. The `VER /R` command is not reliable since it only reports the base version level, not the fix-pack version level, in some releases.

The directory structure in the example above would be represented by the following entries:

```
ga21:6.514;OS/2 2.1 General Availability
ga21mr1:6.617:OS/2 2.11 MR1
warp:8.162:Warp
warp_fp:8.200:Warp Full Pack
```

2.4 PMDF Menus and Options

PMDF offers a number of facilities from its pull-down menus and also from the mouse buttons.

From the Keyboard Ctrl-C and Esc serve to interrupt the Dump Formatter.

Warning

Do not use the Dump Formatter Q command. Under PMDF this will cause PMDF to hang. To terminate the Dump Formatter either quit PMDF from the system menu or select another dump for processing.

The PMDF screen appears as follows:

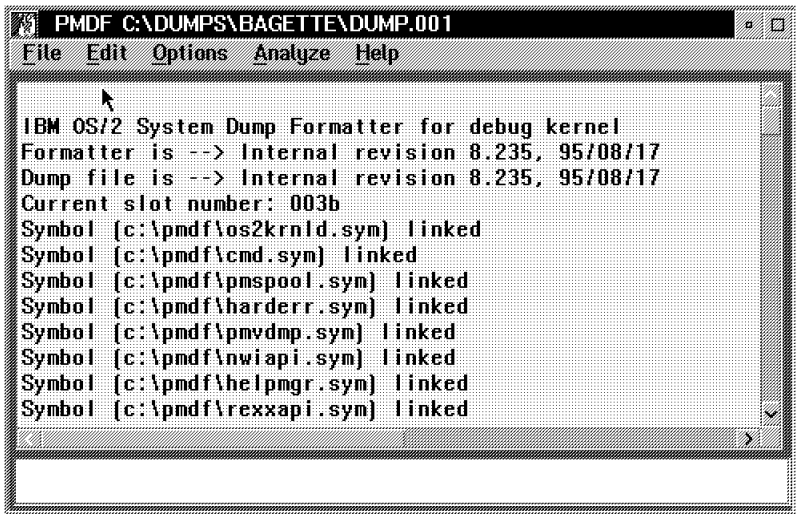


Figure 7. Presentation Manager Dump Formatter

2.4.1 PMDF File Menu

The File pull-down menu offers the following options:

New Dump

Select this option to decompress a new dump.

Notes: For diskette dumps the *DUMPDATA.nnn* files may be copied for a directory on the hard drive and decompressed from there.

PMDF has the ability to decompress diskette images created by OS2IMAGE without re-creating the original diskettes. To use this facility each of the image file must be named *image.nnn* where nnn is a numeric sequence number that corresponds to the disk number.

Open Dump

This option prompts the user for the dump file name and then invokes Dump Formatter.

Log Output

This option prompts the user to start or stop logging output to a file. Data may be appended to an existing log file.

Save Output
This option allows the user to save all output displayed in the PMDF scrollable window.

Connect
Connect allows PMDF to be used as a terminal emulator to drive a Kernel Debugger session. See Chapter 1, “Kernel Debugger User Guide” on page 1 for more information.

Disconnect
Disconnect terminated the communications session with the Kernel Debugger.

The following diagram illustrates the File pull-down menu options.

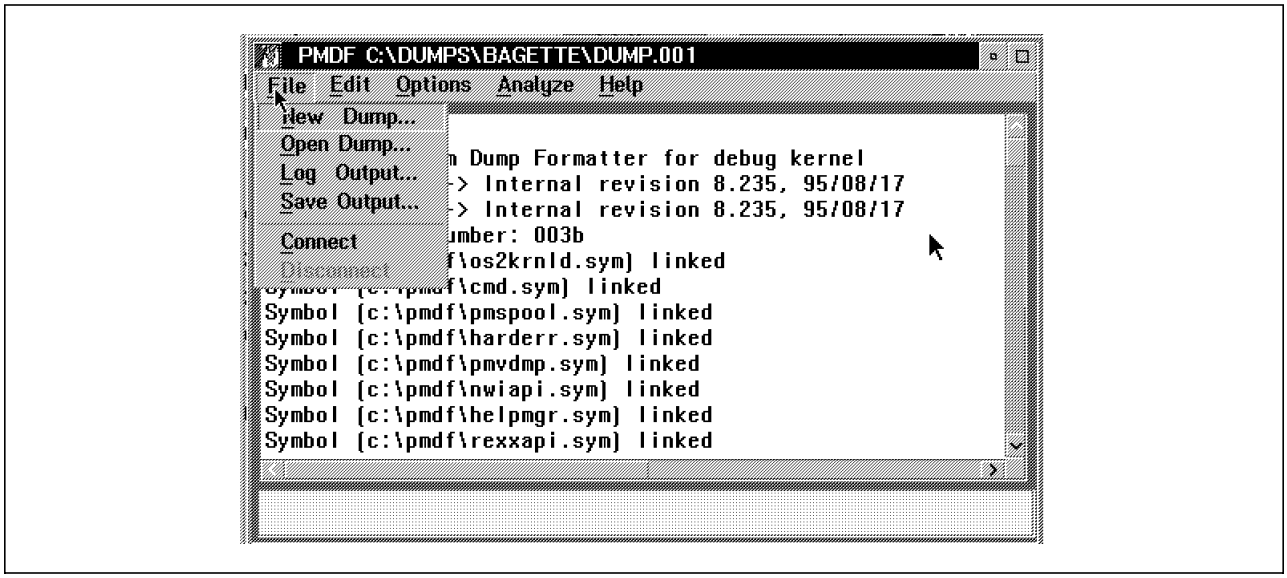


Figure 8. PMDF File Pull-Down Menu

2.4.2 PMDF Edit Menu

The Edit pull-down menu offers the following options:

Search String
Locates text within the scrollable window.

Undo
Reverse the previous Edit Cut action.

Copy
Copy marked text to the clipboard.

Cut
Move marked text to the clipboard.

Clear Screen
Clears the scrollable window of all text. This is not a reversible action.

The following diagram illustrates the Edit pull-down menu options.

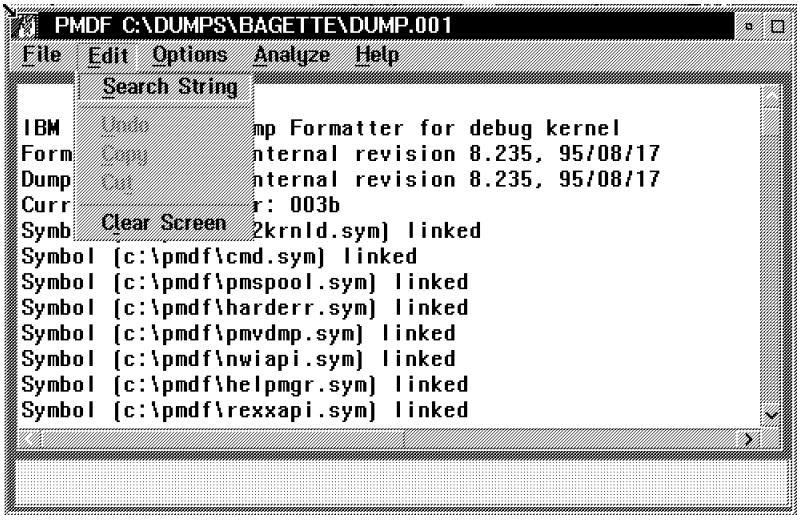


Figure 9. PMDF Edit Pull-down Menu

2.4.3 PMDF Options Menu

The Options pull-down menu offers the following options:

Font Settings
This allows font selection for displayed output.

Function Keys
This provides a menu to predefine function keys as strings of Dump Formatter command strings. Commands may be separated by a semicolon.

Terminal Settings
Allows the communications parameters to be specified for when the Connect option of the File pull-down is selected.

Save Settings
This will save the current options in the PMDF.INI file for use the next time PMDF is started.

The following diagram illustrates the Options pull-down menu options.

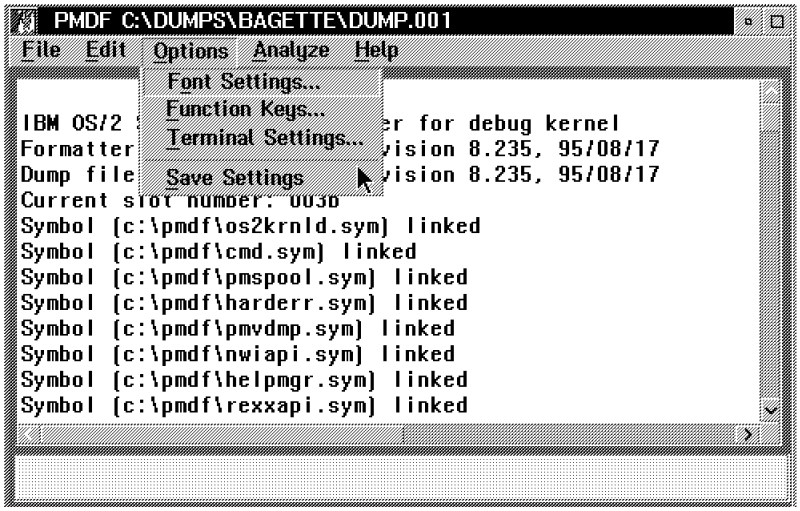


Figure 10. PMDF Options Pull-Down Menu

2.4.4 PMDF Analyze Menu

The Analyze pull-down menu offers four selections, each of which displays its own menu selection. Where parameters are required they should be highlighted by double-clicking mouse button 1 on text in the scrollable window.

Caution

The output from the Analyze options needs to interpreted with care. Some options are precise in that the follow control block chains anchored from the SAS such as the Physical Device Driver Chain and Kernel Heap. Others depend on correct symbols being loaded for correct results. Some options, for example those that display stacks, are more speculative in what they display.

Before these facilities are relied on, the user should thoroughly acquaint themselves with the manual techniques that belie their function. This information is available in the course materials that comprise the first few chapters of this handbook.

The following selections are available:

System

The System menu displays the following options:

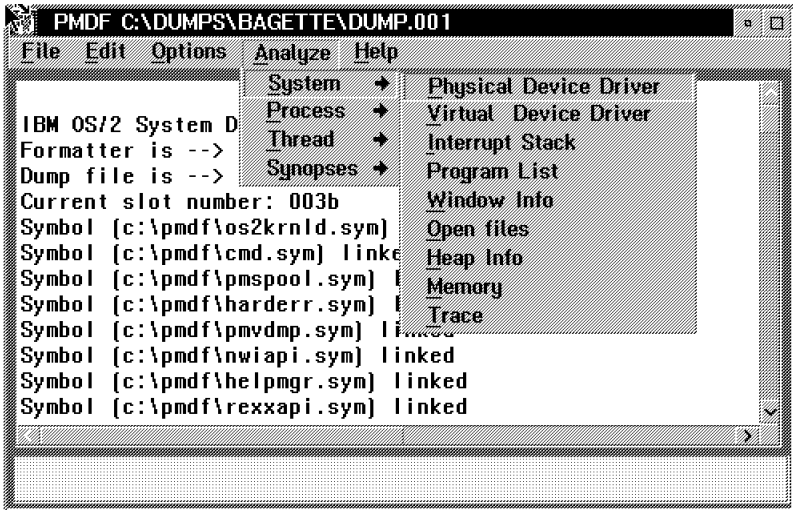


Figure 11. PMDF System Menu

Process

The Process menu displays the following options:

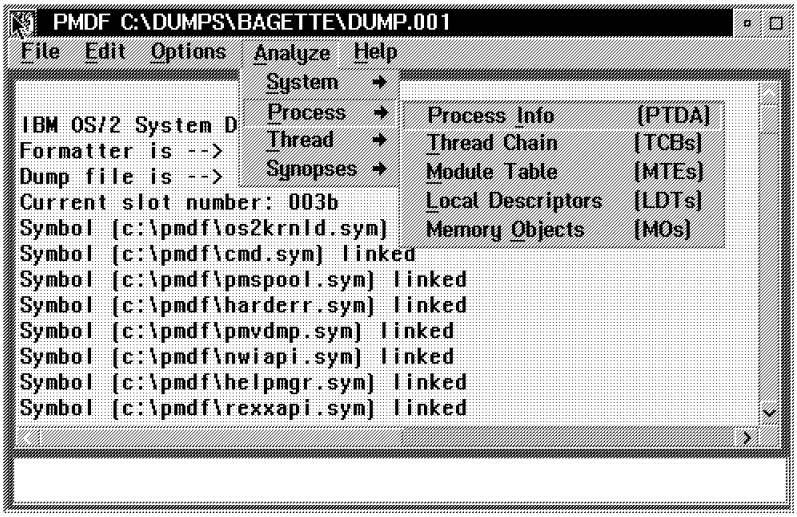


Figure 12. PMDF Process Menu

Threads

The Threads menu dumps stacks related to a given thread. The following menu is displayed:

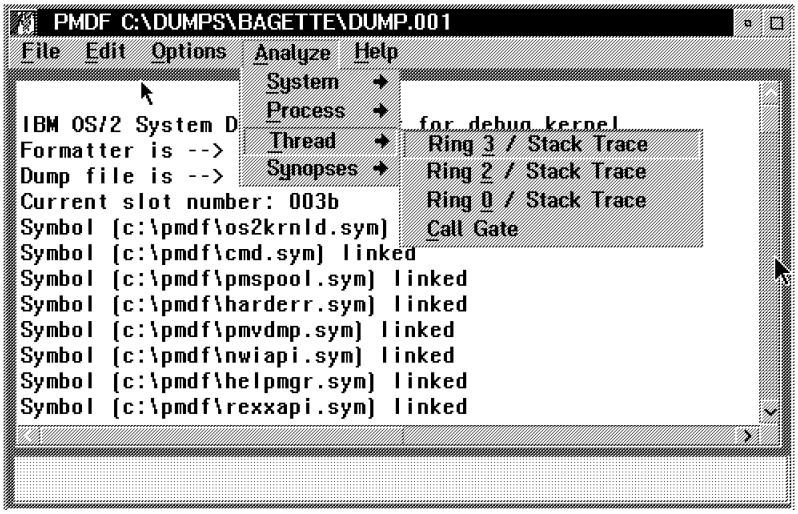


Figure 13. PMDF Threads Menu

Synopsis

This offers a miscellaneous collection of options, the most important of which is the Trap Screen display. The following menu is displayed:

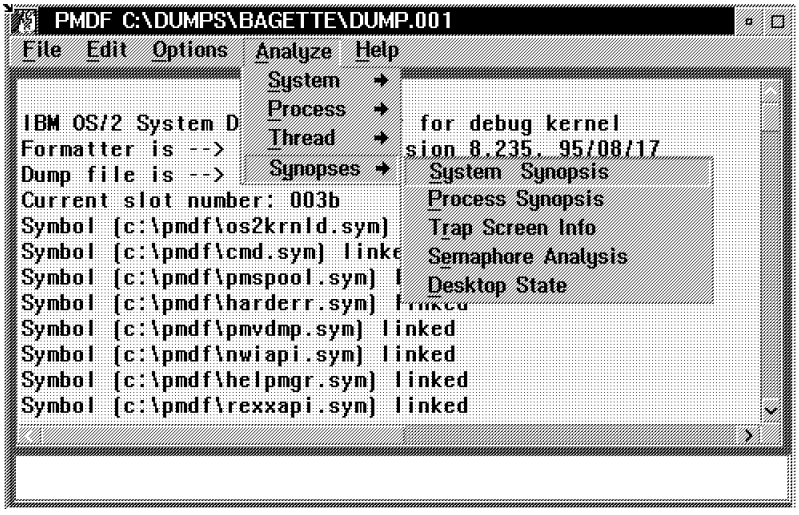


Figure 14. PMDF Synopsis Menu

2.4.5 PMDF Help Menu

The Help pull-down menu offers standard help facilities.

The following diagram illustrates the Help pull-down menu options.

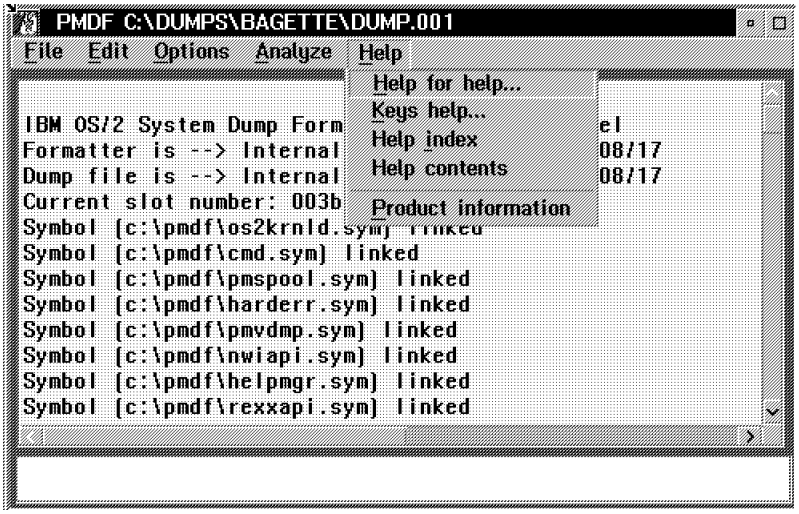


Figure 15. PMDF Help Pull-down Menu

2.4.6 PMDF Mouse Options

Standard CUA mouse selection and highlighting are implemented. Marked items may be dragged and dropped onto the command line.

A double-click with mouse button 1 will highlight a blank delimited string.

A single click with mouse button 2 will display a pop-up menu whose items take the highlighted text in the scrollable output window as input.

The following diagram shows an example of the mouse pop-up menu. In this example the Structures option is displayed. This particular option acts as a supplement to the Dump Formatter `.D command`. For it to work correctly, the Structure Definition Files (*.SDF) are required to be present in the same directory as the Dump Formatter. These files are build level dependent and will only display correct information if matched to the dump level. There is no validation performed by these displays. The user must ensure that an appropriate input address is highlighted.

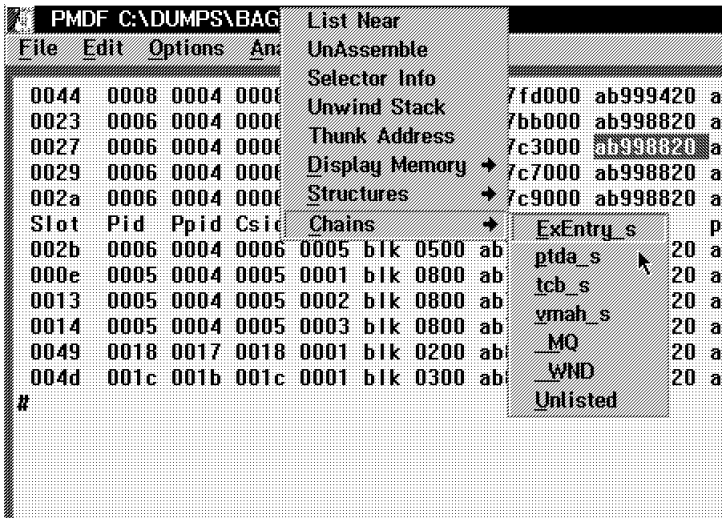


Figure 16. PMDF Address Highlighted

2.5 PMDF REXX Interface

PMDF provides a REXX interface that allows REXX EXECs to issue Dump Formatter commands and capture their output in REXX variables. EXECs are able to display output on PMDF's scrollable output window.

EXECs are invoked by entering the REXX execname, with optional directory information, prefixed with a '%' character from the PMDF command window. If the EXEC is not installed in a directory in the PATH or in the same directory as PMDF, then it must be prefixed with the fully qualified path name. For example:

```
%SEGTAB 123

%C:\MYEXECS\TEST1 parm1 parm2
```

It is also possible to use relative path expressions thus:

```
%..SEGATB 123
```

If a path has to be specified when passing an exec name as a parameter to another exec then quotation marks around the path and file name will be required.

PMDF implements its interface to the Dump Formatter by creating a REXX subcommand environment. The REXX *address* instruction allows an EXEC to execute and capture the output from a Dump Formatter command by addressing this subcommand environment.

The syntax and parameters for this implementation of the **address** instruction are:

```
address df 'CMD' <output> <df_cmd>
```

Where:

<output>
is the name of a stem to a REXX compound variable that will be assigned to capture output from the Dump Formatter command.

″output.0″ will be set to the number of lines. ″output.n″ will contain the nth line of output.

<df_cmd>
is the dump formatter command and parameters.

Parameters following the EXEC name will be passed to the EXEC as a one parameter string.

A number of general purpose EXECs are provided in the OS2PDP package on the CD-ROM accompanying this book. These are:

RUNCHANIN Generalized Control Chain Running EXEC

PS Generalized EXEC for executing Dump Formatter commands per thread slot

TEMPLATE A Template EXEC containing a collection of subroutines useful for writing other EXECs

There are also a number of example EXECs that format control blocks and illustrate how to use the REXX interface and the subroutines contained in TEMPLATE.

2.5.1 The RUNCHAIN EXEC

Syntax

```
RUNCHAIN <addr> link(<offset>,<s>) stopvalue(<stop>) chain(<nnn>) exec(<cmd>)
         print(<file>)
```

This exec provides a generalized control block chaining facility, where at each hop of the chain a command or exec may be executed. The starting address and link offset are required. Other parameters are optional. The parameters to RUNCHAIN are:

<addr> Is an address expression of the start of the chain

<offset> Specifies the decimal or hexadecimal offset of the linking address. Default is 0

<s> Specifies the length of the linking field as: D (double) or W (word) - Default is D

<stop> Specifies a termination value for the linking field. This take precedence over <chain> and may be specified as a hexadecimal or decimal value.

<nnn> Specifies the maximum number of chain hops to traverse. Default is 10

<cmd> Specified a command to be executed at each hop. If the command is prefixed with a % then an exec is executed. @L will cause the linear address of the current block to be substituted. Default is DD @L L4.

<file> Specifies a print file to which the output will be copied.

Note: Hexadecimal values are specified as ′nn′x

As an example, suppose the linear address of an MTE is %fff2bde0. MTEs are linked at +c in os2 2.1. To run the chain of MTEs displaying 8 double words do the following:

```
%RUNCHAIN %fff2bde0 link(c) exec(DB @L L40)
```

The resulting output would be appear thus:

```
Block 1 at %FFF2BDE0

%fff2bde0  00060002 fff2bdfc fff2bfc3 fe0a1dac
%fff2bdf0  0000b980 00000000 00010000 00000000

Block 2 at %fe0a1dac

%fe0a1dac  02600002 fcace908 fe0a1dfc fe083e40
%fe0a1dbc  4498b1c6 0000000d 0000003a fe0addf0

Block 3 at %fe083e40

%fe083e40  024e0002 fcac52f0 fe083e70 fe0adef8
%fe083e50  4498b1c6 00000005 00000038 fdf40fac

Block 4 at %fe0adef8

%fe0adef8  01aa0002 fcac07a0 fe0adf14 fdf61cc8
%fe0adf08  0498b1c8 00000000 00000035 4d495405

Block 5 at %fdf61cc8

%fdf61cc8  01a80002 fca9ad58 fdf61ce4 fdf61d68
%fdf61cd8  0498b1c8 00000000 00000036 53595307
Chain run successfully for 5 hops
```

To format the first 40 MTEs in the chain enter the following:

```
%RUNCHAIN %fff2bde0 link(c) exec(.lmo @L) chain(40)
```

2.5.2 The PS EXEC

Syntax

```
PS <s1> <s2> <cmd> <parms> <;cmd> <parms> .....
```

This is the Per-Slot exec. It will repeatedly execute a DF command string or REXX EXEC for each thread slot in the range specified. The linear addresses of slot related control blocks (TCB, PTDA and TSD) may be specified symbolically in the command string so that the correct address will be substituted for each slot traversed by PS.

The parameters to PS are:

< s 1 > Starting (hexadecimal) slot number

< s 2 > Ending (hexadecimal) slot number or *, which signifies highest active slot in the system.

< c m d > Is any string of DF commands separated by a semicolon ';' or a single REXX EXEC prefixed by %.

<parms> Are any valid parms where @TCB, @PTDA and @TSD are substituted with their corresponding linear addresses. @disp is the scheduler's ESP relative to the TSD. N.B @disp is only defined when page table entries are present for the TSD.

Example 1:

Display priority information (on a 2.11 system) for slots 30 to 33 where priority class is at TCB+e4, priority delta is at TCB+e5 and dispatching priority is a word at TCB+e8.

Enter:

```
%PS 30 33 DB @TCB+e4 L2; DW @TCB+e8 L1

Slot 30
Warning: not all addresses are present
DB %7BA8FE78+E4 L2; DW %7BA8FE78+E8 L1

%7ba8ff5c 02 0f                                          ..
%7ba8ff60  020f

Slot 31
DB %7BA9002C+E4 L2; DW %7BA9002C+E8 L1

%7ba90110 02 00                                          ..
%7ba90114  0200

Slot 32
Warning: not all addresses are present
DB %7BA9002C+E4 L2; DW %7BA9002C+E8 L1

%7ba90110 02 00                                          ..
%7ba90114  0200

Slot 33
DB %7BA90394+E4 L2; DW %7BA90394+E8 L1

%7ba90478 03 00                                          ..
%7ba9047c  0800
ps ended rc: 0
```

Note: For slot 30 a warning message is issued because in this instance .s30 gave an error because slot 30 page tables were swapped out.

2.5.3 The TEMPLATE EXEC

TEMPLATE is not intended to be executed. Rather, it is a model for creating new EXECs. It contains a number of generally useful subroutines used in other EXECs.

Currently included in TEMPLATE are the following subroutines:

linaddr <address>

Converts an address expression to a linear address (without the % prefix). If storage cannot be referenced then a null string is returned.

getstor < h > , < a > , < s > , < f >

Retrieve a byte, word or double word from storage. If storage can not be retrieved, then the DF error msg is returned.

< h > Is a dump handle

< a > Is a DF address expression

< s > Is the size specified as: B, W or D

< f > Is the optional output format, which may be specified as C for character, N for decimal or X for hexadecimal string. X is the default.

gethxstr <h>,<a>,<l>

Retrieve a string of hex bytes from storage. If storage can't be retrieved then a null string is returned. The string is returned as a concatenated string of bytes.

< h > Is a dump handle

< a > Is a DF address expression

< l > Is the length of storage to retrieve

getbytes <h>,<a>,<l>

Retrieve a one or more bytes from storage. If storage can't be retrieved then a null string is returned. The string is returned as a string of bytes separated by blanks.

< h > Is a dump handle

< a > Is a DF address expression

< l > Is the length of storage to retrieve

getwords <h>,<a>,<l>

Retrieve a one or more words from storage. If storage can't be retrieved then a null string is returned.

< h > Is a dump handle

< a > Is a DF address expression

< l > Is the length of storage to retrieve

getdwords <h>,<a>,<l>

Retrieve a one or more double words from storage. If storage can't be retrieved then a null string is returned.

< h > Is a dump handle

< a > Is a DF address expression

< l > Is the length of storage to retrieve

getqwords <h>,<a>,<l>

Retrieve a one or more quadruple words from storage. If storage can't be retrieved then a null string is returned.

< h > Is a dump handle

< a > Is a DF address expression

< l > Is the length of storage to retrieve

format <name>,<offset>,<base>,<type>,<desc>
Format a field from a control block and returns the value of the field.

<name> Is the field name.

<offset> Is a relative hex offset (prefix with + or -)

<base> Is the base address of the control block

<type> Is the filed type (b=byte, w=word, d=double word)

<desc> Is a description of the filed.

fmtblock <name>,<offset>,<base>,<type>,<number>,<desc>
Formats a table of bytes, words or double words imbedded in a control block.

<name> Is the field name.

<offset> Is a relative hex offset (prefix with + or -)

<base> Is the base address of the control block

<type> Is the filed type (b=byte, w=word, d=double word)

<number> Is the number of entries in the table

<desc> Is a description of the filed.

2.6 Process Dump Formatter

PMDF provides a process Dump Formatter facility which is invoked automatically when the Open option of the File pull-down menu is selected against a Process Dump.

The Process Dump Formatter offers a limited subset of the full Dump Formatter command set. These are:

.D Display storage in Bytes, Words or Double-Words.

DL Display LDT entries.

L List, Symbols, Maps and Symbol Groups

.LM and .LMO Display Module Table Entries and Object Tables

.MA Display Arena Records for storage dumped.

.MO Display Object Records for storage dumped.

.ML Display Information on Dumped Memory.

Note: This command does not perform the same function as the similarly named Kernel Debugger **.**ML command, which formats VM Alias Records.

.P Display threads.

Note: Unlike the Dump Formatter and Kernel Debugger version of this command, **.**P is used to select the thread ordinal within the dumped process. Thus for single thread processes **.**P 1 is the only valid combination.

.PB Display thread Block IDs.

Note: Unlike the Dump Formatter and Kernel Debugger version of this command, **.**PB is used to select the thread ordinal within

the dumped process. Thus for single thread processes **.PB 1** is the only valid combination.

R Display registers for each thread.

.S Set default thread slot.

Note: Unlike the Dump Formatter and Kernel Debugger version of this command, **.S** is used to select the thread ordinal within the dumped process. Thus for single thread processes **.S 1** is the only valid combination.

W Load and Unload Symbol files

? Syntax help for internal commands

.? Syntax help for external (dot) commands.

Note: Except where noted above, the command set for the Process Dump Formatter does not support any of the optional parameters supported by their equivalent Kernel Debugger commands.

When a Process Dump is loaded, PMDF displays the following screen:

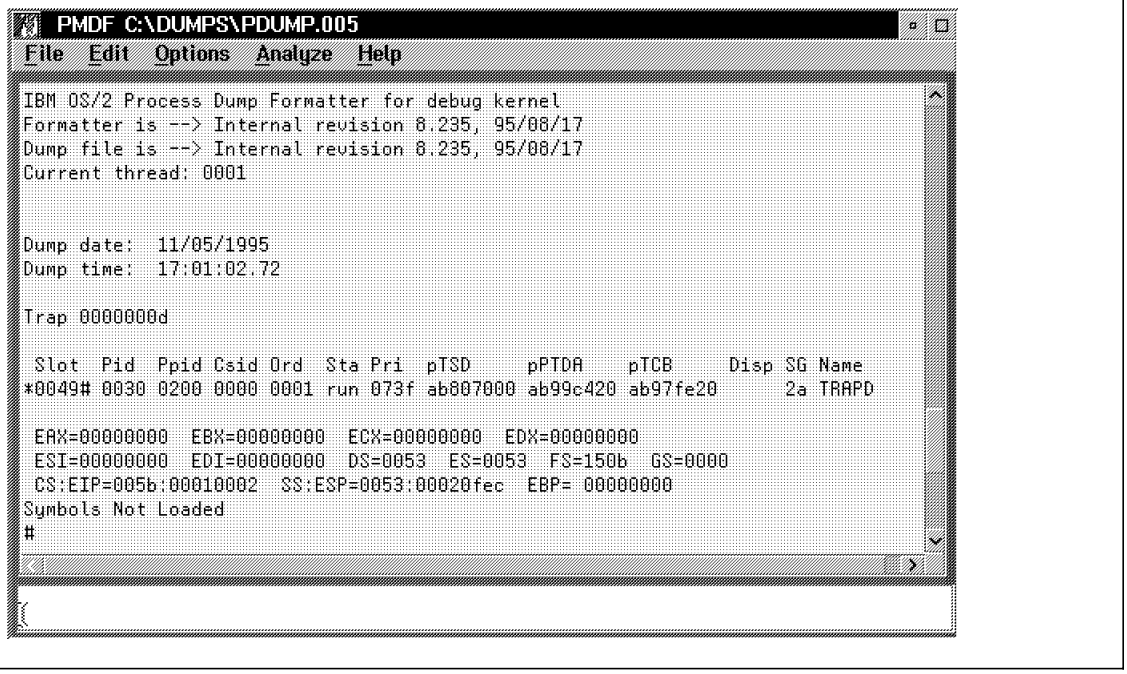


Figure 17. Process Dump Loaded PMDF Display

Note: The data and time of the dump are displayed.

If the dump was created because of a trap then the trap number is displayed otherwise the trap number is shown as **ffffffff**.

The current thread slot and register are shown last.

The Analyze pull-down menu differs from the standard PMDF Analyze facility. This offers the following choices:

Registers
This performs the R command for each thread dumped.

Task Summary
This performs a .P command followed by an R command for each thread dumped.

Local Descriptors
This performs a DL command.

Virtual Memory Control Blocks
This performs a .MA and .MO command.

Module Table
This is a much more extensive version of the .LMO command. The entire MTE and SMTE for each module dumped is formatted.

Process Synopsis
This formats the entire Process Dump, including dumping all memory in byte format.

The Analyze option menu appears as follows:

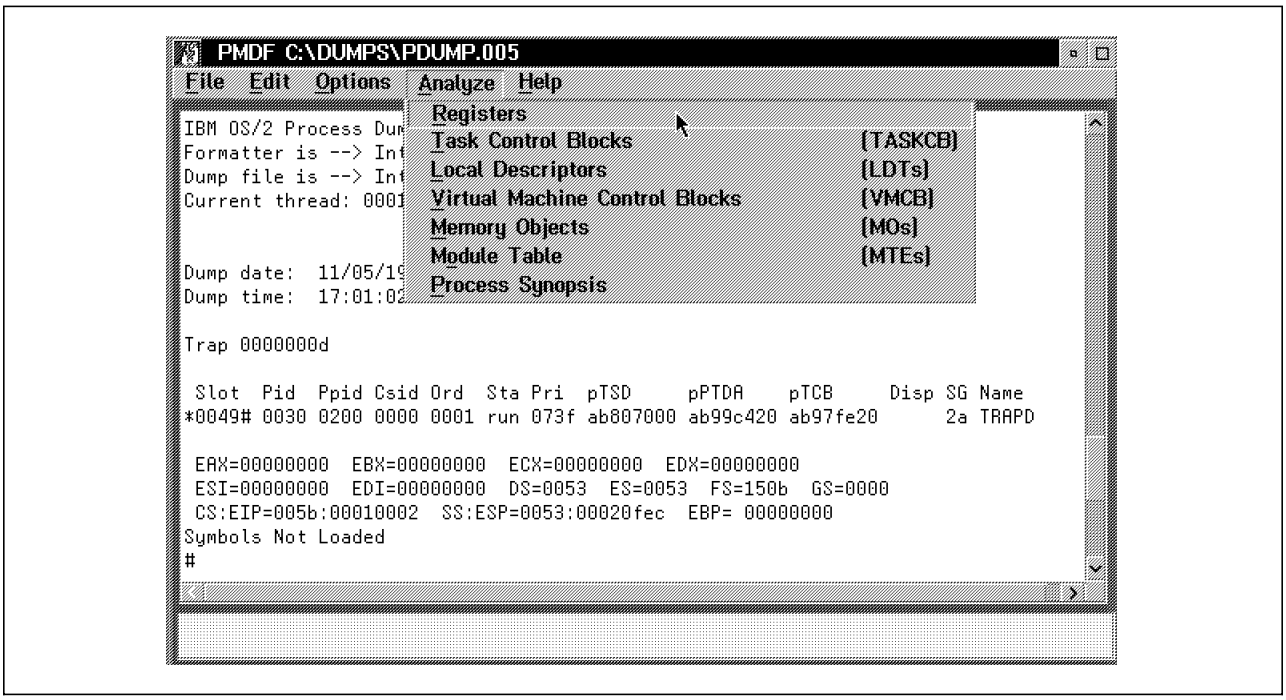


Figure 18. Analyze Option Menu of the PMDF

For information on taking and controlling Process Dumps see the following:

The CONFIG.SYS DUMPPROCESS command

The DosProcessDump API

Chapter 3. Kernel Debugger and Dump Formatter Command Reference

The Kernel Debugger and Dump Formatter share a common subset of commands, which comprises the vast majority of the combined command set. The following symbols will be used to denote to which tool commands are applicable:

Dump Formatter

Kernel Debugger

References to some system control offsets blocks are made in the descriptions of the commands. For the sake of brevity, the ALLSTRICT version of the OS/2 WARP 3.0 kernel is assumed and in some cases the equivalent RETAIL and HSTRICT kernel offsets are given in parentheses. For example:

```
JFN_pTable (PTDA +0x5b8 (H/R: +0x5b0))
```

The reader should refer to the *System Reference Manual, Volume 4* of the OS/2 Debugging Library for control block layouts of the ALLSTRICT, HSTRICT and RETAIL kernels for OS/2 WARP 3.0 and OS/2 2.11 kernels.

Commands are categorized into two classes:

Internal Internal commands begin with an alphabetic character. They are control program independent in the sense that they relate only to the Intel 80x86 hardware architecture.

External External commands are prefixed with a period. They relate to the software environment under analysis and are dependent on the data structures of the operating system environment.

For a description of the conventions used in the syntax diagrams, see 3.1, “Syntax Diagrams - Notation.”

Complex expressions may be used where substituted values are required. The rules governing expressions are described in 3.2, “The Expression Evaluator” on page 73.

3.1 Syntax Diagrams - Notation

The command syntax descriptions for the Dump Formatter and Kernel Debugger use a graphical notation, which is now in common use. The diagrams should be read as a *road map* starting at the sign:

and ending at the sign:

The command verb and options are shown in uppercase type.

Parameter values to be supplied by the user are shown in lowercase. The rules governing the use of complex expressions are described under 3.2, “The Expression Evaluator” on page 73.

Continuation of the syntax diagram is shown by:

→

at the break, and:

►–

at the beginning of the continuation line.

Expansion of syntax into detailed subsections is indicated as follows:

```
─┤ section name ├─
```

Expanded section begins:

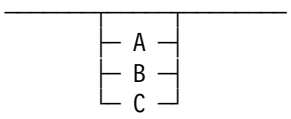
```
section name:
├─
```

Expanded section ends:

```
─┤
```

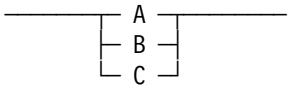
See below for a more detailed description and example of subsections.

Mutually exclusive options where a *non-mandatory* selection is required are shown as follows:

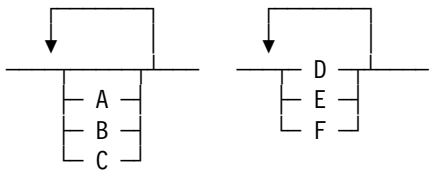


Here at most one of A, B or C may be selected.

Mutually exclusive options where a *mandatory* choice is required are shown as follows:

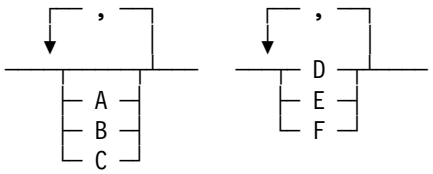


Multiple selections are shown as follows:



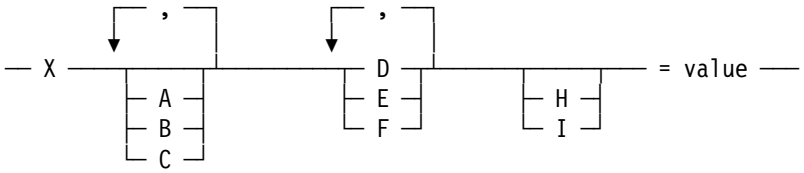
One or more of A, B or C is optional, whereas at least one of D, E or F is required.

If a separator is required between parameters, then it is shown in the diagram. For example, a comma is required between each selection in the following:



For the Dump Formatter and Kernel Debugger spaces between parameters options are optional.

Ordered non-exclusive selection lists and parameters are shown in the order they must be specified.

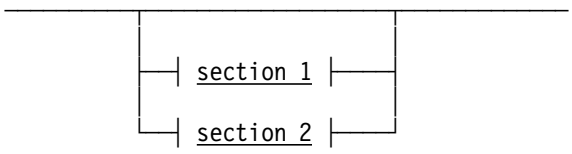


Here, X must be specified first, followed optionally by none, one or more of A, B or C separated by commas, followed by at least one of D, E or F separated by commas, followed optionally by H or I and finally by the character = and a quantity substituted for *value*.

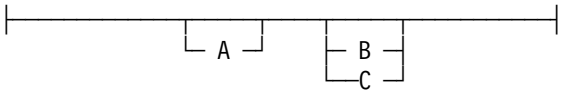
The following examples would be correct interpretations of this last syntax diagram:

```
X D = 55
XA,B I=444
```

Where complex diagrams require splitting into multiple sections, the sections are identified by a lowercase italic name. For example:



section 1:



In this example the syntax for section 1 is exclusive with section 2. The options for section 1 are shown at the label *section 1 :*

3.2 The Expression Evaluator

The Kernel Debugger and Dump Formatter expression evaluator supports a variety of arithmetic, Boolean and addressing operators to form a value to be substituted into a command parameter may be derived. The atomic entities used within expressions may be string or numeric in type. Arithmetic expressions may be used with addressing separators to represent a physical,

linear, selector:offset or segment:offset address. Certain conventional values may be represented in expressions by mnemonics.

Symbols defined by symbol files may also be used to represent either their equivalent address operator and address arithmetic value combination or constant arithmetic value in command line expressions.

3.2.1 String Expressions

These are identified by being enclosed in either single or double quotes. A string may contain any keyboard character including quotation marks, which must be duplicated so as not to act as a string terminator. Examples are:

```
'this is a sting'
'That''s an other example'
"and so is this"
```

Where there is no ambiguity, the terminating quote may be omitted.

3.2.2 Arithmetic Expressions

The expression evaluator will accept numeric values in a decimal, hexadecimal, binary and octal notation. These are indicated as follows:

nnnnnnY
Binary number **nnnnnn**

nnnnnnO
Octal number **nnnnnn**

nnnnnnQ
Alternative notation for octal number **nnnnnn**

nnnnnnT
Decimal number **nnnnnn**

nnnnnnH
Hexadecimal number **nnnnnn**

The base suffix may be in uppercase or lowercase.

The default base when a suffix is omitted is hexadecimal.

The following represents the same number, expressed in each of the permissible forms:

```
31
31t
1fh
37o
37q
10001111y
```

Arithmetic expressions are of the following three types:

Absolute An arithmetic expression that resolves to a numeric value.

Absolute expressions may be formed from numeric values using arithmetic binary and unary operators and built-in functions together with parentheses (), to influence evaluation order.

Boolean Boolean expressions are ones that resolve to either a TRUE or FALSE value.

Boolean expression may be formed from arithmetic expressions using boolean binary and unary operators together with parentheses (), to influence evaluation order.

Boolean expressions may be used as absolute values in arithmetic expressions, where TRUE assumes the value 1 and FALSE 0.

Address An arithmetic expression that resolves to one or two numeric values that represent a linear, physical, segment:offset or selector:offset address.

Address expressions are formed from absolute expressions using addressing separators.

Note: The expression evaluator allows arithmetic values to be expressed in hexadecimal. A potential conflict may occur where symbol names exist that begin with letters *a* - *f*. For example, a linear address expressed as *%fe1234* may be rejected with the message:

```
Invalid expression
```

where a symbol *f* or *fe* is defined. To avoid this conflict prefix the hexadecimal numeric value with a zero, as follows:

```
%0fe1234
```

If the same error message persists, then the address refers to either paged out or unallocated virtual memory.

3.2.2.1 Binary Operators

Arithmetic operators:

The following binary operators are permissible in any arithmetic expression:

***** Multiplication

/ Integer division

MOD
Modulo or remainder operator

+ Addition

- Subtraction

AND
Bitwise AND

XOR
Bitwise exclusive OR

OR
Bitwise OR

Boolean operators:

The following binary operators are permissible in any Boolean expression:

> Greater than

< Less than

> =
Greater than or equal to

= =
Logical equality

! = Logical inequality

&&
Logical AND

|| Logical OR

3.2.2.2 Unary Operators

Arithmetic operators:

The following unary operators are permissible in any arithmetic expression:

NOT
Bitwise ones complement

_ Bitwise twos complement

Boolean operators:

The following unary operator is permissible in any Boolean expression:

! Logical negation

3.2.2.3 Built-in Functions

The following built-in functions operate in a single address expression operand:

SEG
Returns the segment or selector portion of an address that resolves to either a *&segment:offset* or *#selector:offset* form.

OFF
Returns the offset of an address the resolves to either a *&segment:offset* or *#selector:offset* form.

BY
Returns one byte from an address location.

WO
Returns one word from an address location.

DW
Returns one doubleword from an address location.

POI
Returns one doubleword far pointer (selector:offset or segment:offset address) from an address location. The low order word returned is treated as the offset. The high order word returned is treated as a selector or segment based, depending on the default addressing mode. See the D command for more information.

PORT
Returns one byte from an 8-bit I/O port address.

WPROT
Returns one word from a 16-bit I/O port address.

Example:

```
DD %(dw(%7abcde0+10))
```

Display the storage whose linear address is at location %7abcdf0.

3.2.2.4 Address Separators and Address Expressions

The following separators may be used with absolute expressions to form elements of an address:

& Segment prefix

Selector prefix

% Linear address prefix

% %
Physical address prefix

: A segment/offset address separator

| Thread slot number qualifier

Where virtual addresses map to different physical addresses in different processes (typically private arena data and shared arena instance data) then | may be used to qualify the address by thread slot number.

Note: This qualifier is ignored by the Dump Formatter.

Examples:

```
%ebp
```

The value of *EBP* assumed to be a linear address.

```
%%10034
```

The physical address at location 10034.

```
38 | #1f:0
```

The selector:offset address 1f:0 in the context of slot 38.

3.2.2.5 Evaluation Order

Expression are evaluated left to right by applying the following order of precedence to operators, separators and built-in functions:

1. ( )
2. | :
3. & # % %% _ ! NOT SEG OFF BY WO DW POI PORT WPORT
4. * / MOD
5. + -
6. > < > = < =

7. == !=
8. AND XOR OR
9. && ||

3.2.2.6 Mnemonics and Symbols

Symbols defined in symbol files may be used in any arithmetic expression. Absolute symbols (that is, symbols of constants) are treated as absolute expressions. Other symbols are treated as address expressions. Symbols are activated using the WA command.

The built-in register mnemonics supported by the Kernel Debugger and Dump Formatter are:

- 16-bit registers:

  ax, bx, cx, dx, si, di, bp, ip, pc

- 32-bit registers:

  eax, ebx, ecx, edx, esi, edi, ebp, eip

- Segment registers:

  cs, ds, es, fs, gs, ss

- Flag registers:

  flg, eflg

- Control registers:

  cr0, cr2, cr3

- GDTR register:

  gdtb, gdtl

- IDTR register:

  idtb, idtl

- Task control registers:

  tr, ldtr, msw

- Debug registers:

  dr0, dr1, dr2, dr3, dr4, dr5, dr6

- Test registers:

  tr6, tr7

These may be used as absolute expressions for the current register value. See the R command for information on displaying and setting current register values and for the definition of the register mnemonics.

The Kernel Debugger also defines mnemonics:

- br0, br1, br2, ..., br9

to represent the addresses of breakpoints defined by the BP and BR commands.

The expression evaluator allows the prefix @ to a symbol name to distinguish it from a similarly named mnemonic name. For example, **@ax** refers to the symbol **ax**, whereas **ax** refers to the **ax** register value.

Similar conflicts may also arise between hexadecimal values and symbols. These may be avoided by prefixing the hexadecimal numeric value with a zero.

3.3 Internal Commands

The following comprise the set of internal commands:

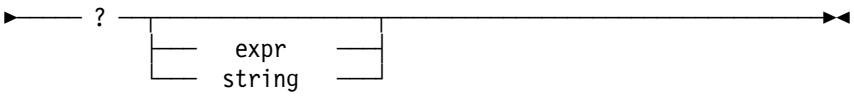
?	Display internal command help
B	Breakpoint command family
BC	Clear breakpoint
BD	Disable breakpoint
BE	Enable breakpoint
BL	List breakpoints
BP	Set or change a breakpoint
BR	Set a debug register breakpoint
BS	Show timestamped breakpoint trace
BT	Set timestamped breakpoint trace
C	Compare memory
D	Dump memory data (default)
DA	Dump memory ASCII data
DB	Dump memory byte data
DD	Dump memory double-word data
DG	Dump global descriptor table
DI	Dump interrupt descriptor table
DL	Dump local descriptor table
DP	Dump Page Tables
DT	Dump Task State Segment
DW	Dump memory word data
DX	Dump 80286 Loadall buffer
E	Enter memory data
F	Fill memory
G	Go
H	Perform hexadecimal arithmetic
I	Input from 16-bit I/O port
J	Execute commands conditionally.
K	Display current stack
.L	List maps, groups and symbols
M	Move memory data
O	Output to 16-bit I/O port
P	Process Trace
Q	Quit the Dump Formatter
R	Set or Display Current CPU Registers

S	Search
T	Trace
U	Unassemble
V	Trap Vectors Command family
W	Withmap Add or Remove
Y	Set Kernel Debugger options
Z	Set/list/execute the default command

3.3.1 ? - Show Internal Command Help or Evaluate an Expression

Display help for internal Kernel Debugger and Dump Formatter commands and evaluate complex expressions.

Syntax:



Parameters:

(default) Displays a help summary for most of the Dump Formatter and Kernel Debugger internal commands.

Note: Some of the information displayed is out-of-date.

Two pages of information are displayed with an intervening **--More--** prompt.

expr An expression that resolves to either a simple numeric value or an address using any of the expression evaluation operators. Symbols of addresses and symbols of absolute values may be specified.

string A string enclosed in single or double quotes.

Results and Notes:

If an expression is specified, then it is evaluated. If it resolves to an address, then it is displayed in equivalent forms, as follows:

```
sel:offset %linaddr %%physaddr
```

Where:

sel:offset Specifies the selector and offset form of the address if the expression resolves to a ***sel:offset*** form.

%linaddr Specifies the linear address equivalent of the expression if it resolves to either a ***sel:offset*** or ***%linaddr*** form.

%physaddr Specifies the physical address equivalent of the expression. If the expression resolves to a virtual address then the page tables must be present to perform the address translation.

See the DP command for information on displaying page table entries and the .I command for information on paging in memory.

If the evaluated expression resolves to an absolute value then it is displayed in hexadecimal, decimal, octal, binary, character and Boolean forms. For example:

```
##? 5
05H 5T 5Q 00000101Y '.'  TRUE
? bmp_segsize
12H 18T 22Q 00010010Y '.'  TRUE
```

Notes

Each arithmetic value is suffixed with a modifier that indicates the base used:

H Signifies hexadecimal

T Signifies decimal (Tens)

Q Signifies octal (Qctal?)

Y Signifies binary (Yes/no?)

In the previous example, *bmp_segsize* is an absolute symbol of value *0x0012* defined in map OS2KRNL.

If a string expression is displayed, then it is echoed back to the console. For example:

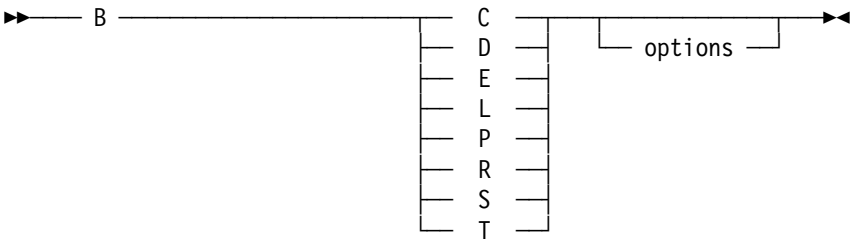
```
##? "This is a way of annotating the debug log from this session's analysis"
This is a way of annotating the debug log from this session's analysis

##
```

Note: Evaluation of simple expressions involving two absolute expressions may be done using the H command.

3.3.2 B - Breakpoint Command Family

The breakpoint family of eight commands provide a means of defining and managing *sticky* breakpoints.

Syntax:



Parameters:

C Clear breakpoints.
See the BC command for options.

D Disable breakpoints.
See the BD command for options.

E Enable breakpoints.
See the BE command for options.

L List breakpoints.
See the BL command for options.

P Set or change breakpoints.
See the BP command for options.

R Set a debug register breakpoint.
See the BR command for options.

S Set a timestamped breakpoint trace.
See the BS command for options.

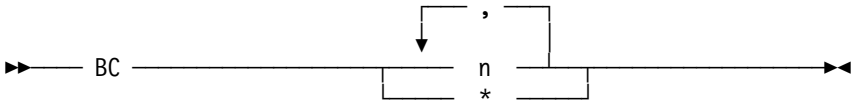
T Display a timestamped breakpoint trace.
See the BT command for options.

options See the associated command for details.

3.3.3 BC - Clear Breakpoints

Clear 1 or more breakpoints.

Syntax:



Parameters:

n Breakpoint number to be cleared.

***** may be specified to clear all breakpoints.

See the breakpoint commands for information on listing and setting breakpoints.

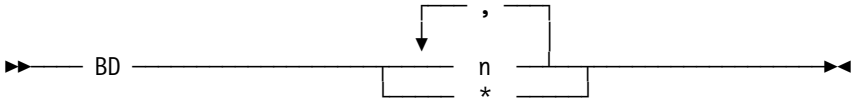
Results and Notes:

The specified breakpoints are cleared. No information is displayed.

3.3.4 BD - Disable Breakpoints

Disable 1 or more breakpoints.

Syntax:



Parameters:

n Breakpoint number to be disabled.

***** may be specified to disable all breakpoints.

See the breakpoint commands for information on listing and setting breakpoints.

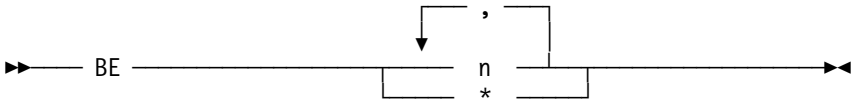
Results and Notes:

The specified breakpoints are disabled. No information is displayed.

3.3.5 BE - Enable Breakpoints

Enable 1 or more breakpoints.

Syntax:



Parameters:

n Breakpoint number to be enabled.

***** may be specified to enable all breakpoints.

See the breakpoint commands for information on listing and setting breakpoints.

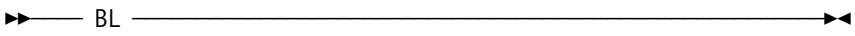
Results and Notes:

The specified breakpoints are enabled. No information is displayed.

3.3.6 BL - List Breakpoints

List all breakpoints defined by the BP and BR commands.

Syntax:



Parameters: none.

Results and Notes:

BL lists the definitions of all currently defined breakpoints. An example of this follows:

```
bl
0 e  0158:00005874 [DOSOPEN]  5 (5) ".P*;G"
1 d  0158:00007384 [DOSHOLDSIGNAL]
2 e  %fff461a4 [_tkSchedNext]  12 (15) ".P*"
3 dT %fff474e4 [_PGSwitchContext]
4 d  %1a022298 [DOS32WRITE]  10 (10)
5 e  W2 0030:0000ffcc [Ppid]
6 dI E1 0000:00000000   5 (5) "DW TASKNUMBER L1"
##
```

Breakpoint definitions are of the following two forms:

1. The general layout of the **BP** breakpoint definition is:

   ```
   n st addr [symbol] pc (mc) "cmd, cmd, ...."
   ```

2. The general layout of the **BR** breakpoint definition is:

   ```
   n st tn addr [symbol] pc (mc) "cmd, cmd, ...."
   ```

Each of the fields has the following meaning:

n The breakpoint number assigned to the given breakpoint.

st The status of the breakpoint:

d Disabled breakpoint. See the BD command.

e Enabled breakpoint. See the BE command.

The suffix **T** signifies that the breakpoint is a timestamp breakpoint created using the BT command

The suffix **I** indicates that the address has become invalid.

addr The address at which the breakpoint is defined.

[**symbol**] The breakpoint offset to the nearest symbolic address, if it exits. See the LN and WA commands for information on listing and loading symbol definitions.

pc The remaining *passcount* for this breakpoint. If a *passcount* is not defined then this value is not displayed. See the BP and BR commands for more information on passcounts.

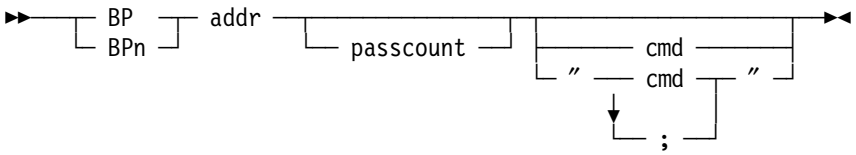
(mc) The initial passcount defined for this breakpoint. If no passcount was defined then this value is not displayed. See BP and BR commands for more information on *passcounts.*

*"***cmd, cmd, ...***"* A list of commands to be executed when the breakpoint *fires.* Each command is separated by commas and the entire string is enclosed in quotes. If no command string is defined then this field is not displayed. See the BP, BR and Z commands for more information on breakpoint command lists.

3.3.7 BP - Set or Alter a Breakpoint

Set or re-specify a software *sticky* breakpoint. by inserting an INT 3 instruction.

Syntax:



Parameters:

n Explicitly specifies a breakpoint number to be assigned to this breakpoint. A value from 0 to 9 may be specified. If specified there must be no space between the number and the BP command.

The default is to assign the lowest available number. If all 10 breakpoint numbers have been assigned then the following message appears:

```
Too many breakpoints
```

addr The address of the breakpoint.

The Kernel Debugger saves the byte of storage at the location specified by ***addr*** and inserts an INT 3 instruction in its place.

Notes: Whenever the Kernel Debugger is entered the storage overlayed by any breakpoints is temporarily restored. When

the Kernel Debugger gives control back to the system, enabled breakpoints are re-instated.

If ***addr*** specifies the address of an existing breakpoint then the existing breakpoint is updated with the new parameters.

Each breakpoint address is recorded with its associated process context. For shared data this is of no consequence. However for private addresses, especially those in the private arena the ***addr*** may be qualified by slot number by using the | operator. This acts as a shorthand to save changing contexts using the .S command in order to set the breakpoint correctly. For example, suppose the current slot is 8, then:

```
BP 31 | %10032
```

Is equivalent to:

```
.S 31
BP 31 | %10032
.S 8
```

passcount Specifies the number of times the breakpoint may be passed before the Kernel Debugger is entered. Each time the breakpoint is passed the count is decremented by 1 until 1 is reached. When the breakpoint is encountered with a count of 1 then it will fire and the Kernel Debugger will be entered. Thus if *passcount*is 5 then the breakpoint will fire on the 5th encounter.

The default passcount is 1, that is, the breakpoint will fire on first encounter.

cmd Specifies a command to be executed when the breakpoint fires. More than one command may be specified by using a semicolon separator and enclosing the entire command list in single or double quotes.

If no command string is specified then the default command string, as specified by the Z command will be executed.

Results and Notes:

If the specified address is valid then the breakpoint definition is accepted. Otherwise one of the following messages is generated:

```
Invalid linear address: %nnnnnnnn
Invalid selector: selector:offset
Past end of segment selector:offset
```

If the breakpoint is successfully defined then the built-in mnemonic *BRn*, where *n* corresponds to the breakpoint number, takes the value of the breakpoint address. This may be used in any address expression or any command.

Note: Since *BP* breakpoints are implemented by the insertion of INT 3 instructions, it is possible for such breakpoints to become discarded if the page of code is discarded and subsequently paged back into memory.

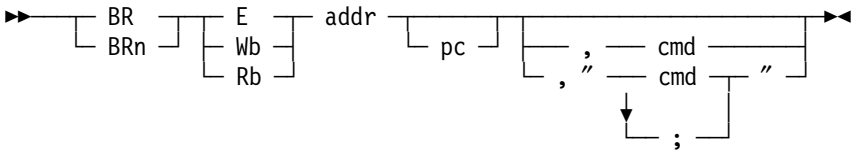
If the **.**I command is used to swap in a page of code, then the breakpoints are automatically restored. (In earlier versions of OS/2 it was necessary to specify the **B** option of **.I**).

This complexity may be avoided by setting register breakpoints with the BR command.

3.3.8 BR - Set or Alter a Debug Register Breakpoint

Set or alter a *sticky* breakpoint, using the debug registers.

Syntax:



Parameters:

n Explicitly specifies a breakpoint number to be assigned to this breakpoint. A value from 0 to 9 may be specified but from this range only a total of 4 may specify enabled debug register breakpoints.

If a value ***n*** is specified there must be no space between the number and the BR command.

The default is to assign the lowest available number. If all 10 breakpoint numbers have been assigned then the following message appears:

```
Too many breakpoints
```

If all four debug registers are in use, then the following message:

```
Out of debug registers
```

is displayed.

Note: A disabled debug register breakpoint does not commit the use of a debug register. Thus more that 4 debug register breakpoints may be defined, but only a maximum of 4 enabled at any one time.

See the BE and BD commands for information on enabling and disabling breakpoints.

E Specifies that the breakpoint is to fire when an instruction at the breakpoint address is fetched for execution.

This is mutually exclusive with the **W** and **R** parameters.

Rb Specifies that the breakpoint is to fire when storage at the breakpoint address, for length ***b*** is referenced. ***b*** may specify 1, 2 or 4 bytes and defaults to 1 byte if left blank.

This is mutually exclusive with the **W** and **E** parameters.

Wb Specifies that the breakpoint is to fire when storage at the breakpoint address, for length ***b*** is stored. ***b*** may specify 1, 2 or 4 bytes and defaults to 1 byte if left blank.

This is mutually exclusive with the **R** and **E** parameters.

addr The address of the breakpoint.

The Kernel Debugger converts the address to a linear address before setting up the debug registers. If the address is invalid the definition is retained but marked disabled and invalid.

Note: Real addresses may not be used with debug register breakpoints.

passcount Specifies the number of times the breakpoint may be passed before the Kernel Debugger is entered. Each time the breakpoint is passed the count is decremented by 1 until 1 is reached. When the breakpoint is encountered with a count of 1 then it will fire and the Kernel Debugger will be entered. Thus if *passcount* is 5 then the breakpoint will fire on the 5th encounter.

The default passcount is 1, that is the breakpoint will fire on first encounter.

cmd Specifies a command to be executed when the breakpoint fires. More than one command may be specified by using a semicolon separator and enclosing the entire command list in single or double quotes.

If no command string is specified then the default command string, as specified by the Z command will be executed.

Note: The command list must be preceded by a comma, unlike the BP command where the comma is optional.

Results and Notes:

If the specified address is valid then the breakpoint definition is accepted and enabled. Otherwise it is accepted but disabled and one of the following messages is generated:

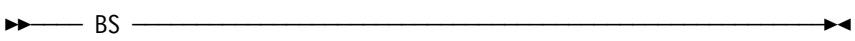
```
Invalid selector: selector:offset
Past end of segment selector:offset
```

If the breakpoint is successfully defined then the built-in mnemonic **BRn**, where **n** corresponds to the breakpoint number, takes the value of the breakpoint address. This may be used in any address expression or any command.

3.3.9 BS - Show Timestamped Breakpoint Trace

Show the timestamped breakpoint trace.

Syntax:



Parameters:

None.

Results and Notes:

The timestamp trace buffer is formatted in LIFO order. The following is an example of the formatted trace:

```
Number of entries = 4284
BP0 381e6ae1a (hex)
BP4 381e68292 (hex)
BP0 381e658d1 (hex)
BP4 381e40559 (hex)
BP0 381e3da7d (hex)
```

Notes: The number of entries is the total accumulated number of timestamp trace events regardless of wrapping of the (4096 entry) timestamp trace buffer.

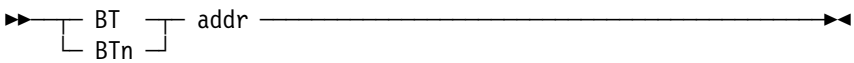
Each entry show the breakpoint number that corresponds to the timestamped breakpoint that fired, the high resolution time stamp in microseconds and a reminder that this value is in hexadecimal.

For information on defining a timestamp breakpoint see the BT command.

3.3.10 BT - Set Timestamped Breakpoint Trace

Set a timestamped breakpoint trace.

Syntax:



Parameters:

n Explicitly specifies a breakpoint number to be assigned to this breakpoint. A value from 0 to 9 may be specified. If specified, there must be no space between the number and the BT command.

The default is to assign the lowest available number. If all 10 breakpoint numbers have been assigned then the following message appears:

```
Too many breakpoints
```

addr The address of the breakpoint.

The Kernel Debugger saves the byte of storage at the location specified by ***addr*** and inserts an **INT 3** instruction in its place.

Notes: Whenever the Kernel Debugger is entered the storage overlayed by any breakpoints is temporarily restored. When the Kernel Debugger gives control back to the system, enabled breakpoints are re-instated.

If ***addr*** specifies the address of an existing breakpoint then the existing breakpoint is updated with the new parameters.

Results and Notes:

If the address is valid then the breakpoint definition is accepted and enabled. When enabled, the timestamp breakpoint causes the current high resolution

system time to be saved in a timestamp circular trace buffer whenever the breakpoint address is executed.

The trace buffer will record up to 4K of entries before wrapping.

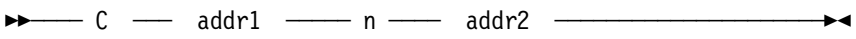
Unlike the BP and BR commands, BT does not return control to user when the breakpoint is encountered.

The timestamp trace may be displayed using the BS command.

3.3.11 C - Compare Memory

Compare up to 64K bytes of memory at two locations in storage.

Syntax:



Parameters:

addr1 The address of the beginning of the first location to compare with the second. This address is assumed to be in ***#selector:offset*** format. If the selector is omitted then the current **DS** selector is assumed.

n The offset from ***addr1*** of the last byte to compare (that is, the length of the range less 1).

addr2 The address of the beginning of the second location to compare with the first. An address expression may be specified. This address is assumed to be in ***#selector:offset*** format. If the selector is omitted then the current **DS** selector is assumed.

Results and Notes:

Storage is compared, if no differences are found then the command prompt is displayed. If either of the addresses is invalid then an error message is displayed.

Where differences are found in the address range, they are displayed in the following way:

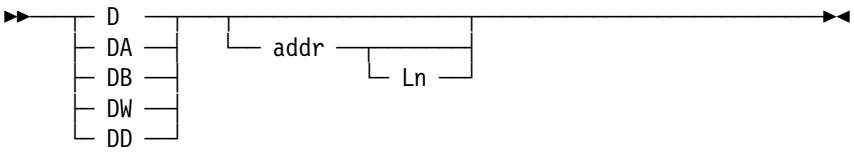
```
001f:00000000 57 4f 001f:00000003
001f:00000001 50 32 001f:00000004
001f:00000003 57 4f 001f:00000006
```

The addresses of the two differing locations are displayed outermost and the bytes at those locations are displayed in columns 2 and 3.

3.3.12 D - Display Memory

Display a range of memory from a given address.

Syntax:



Parameters:

(default) Display memory using the current display format. When the user breaks into the Kernel Debugger the current format is set to byte display. If the user subsequently executes a **DW**, **DA** or **DD** command then the current format is set to words, ASCII or doublewords, respectively. Byte format default may be restored by using DB.

A Force memory to be displayed in ASCII format and set the current display format to ASCII. The display is terminated as soon as the first null byte (**0x00**) is reached or the length specification is reached.

Note: The current display address is not updated when in ASCII format.

B Force memory to be displayed in byte format and set the current display format to byte.

W Force memory to be displayed in word format and set the current display format to word.

D Force memory to be displayed in doubleword format and set the current display format to doubleword.

addr The address of the memory location to display. When the user breaks into the Kernel Debugger this defaults the current **DS** selector, offset **0**. If a display command other than DA is executed then the current display address is updated to the last displayed address **+ 1** .

An address expression may be specified.

Ln The number of bytes, words or doublewords to display, depending upon the current display format. If not specified this defaults to **128** bytes, **64** words and **32** doublewords respectively.

Results and Notes:

Memory is displayed according to the selected display format providing the address is valid. If it is not, but the address represents pageable storage then this may be paged in to memory using the .I command.

The following examples show output in the four different formats.

```
##da 1f:0

001f:00000000 WP_OBJHANDLE=177110
```

Figure 19. ASCII Format

```
##db 1f:0

001f:00000000 57 50 5f 4f 42 4a 48 41-4e 44 4c 45 3d 31 37 37 WP_OBJHANDLE=177
001f:00000010 31 31 30 00 55 53 45 52-5f 49 4e 49 3d 43 3a 5c 110.USER_INI=C:\
001f:00000020 4f 53 32 5c 4f 53 32 2e-49 4e 49 00 53 59 53 54 OS2\OS2.INI.SYST
001f:00000030 45 4d 5f 49 4e 49 3d 43-3a 5c 4f 53 32 5c 4f 53 EM_INI=C:\OS2\OS
001f:00000040 32 53 59 53 2e 49 4e 49-00 4f 53 32 5f 53 48 45 2SYS.INI.OS2_SHE
001f:00000050 4c 4c 3d 43 3a 5c 4f 53-32 5c 43 4d 44 2e 45 58 LL=C:\OS2\CMD.EX
001f:00000060 45 00 41 55 54 4f 53 54-41 52 54 3d 54 41 53 4b E.AUTOSTART=TASK
001f:00000070 4c 49 53 54 2c 46 4f 4c-44 45 52 53 00 52 45 53 LIST,FOLDERS.RES
```

Figure 20. Byte Format

```
##dw 1F:0

001f:00000000  5057 4f5f 4a42 4148 444e 454c 313d 3737
001f:00000010  3131 0030 3555 5245 495f 494e 433d 5c3a
001f:00000020  534f 5c32 534f 2e32 4e49 0049 5953 5453
001f:00000030  4d45 495f 494e 433d 5c3a 534f 5c32 534f
001f:00000040  5332 5359 492e 494e 4f00 3253 535f 4548
001f:00000050  4c4c 433d 5c3a 534f 5c32 4d43 2e44 5845
001f:00000060  0045 5541 4f54 5453 5241 3d54 4154 4b53
001f:00000070  494c 5453 462c 4c4f 4544 5352 5200 5345
```

Figure 21. Word Format

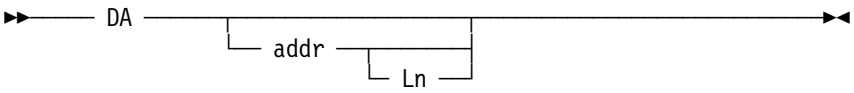
```
##dd 1f:0

001f:00000000  4f5f5057 41484a42 454c444e 3737313d
001f:00000010  00303131 52455355 494e495f 5c3a433d
001f:00000020  5c32534f 2e32534f 00494e49 54535953
001f:00000030  495f4d45 433d494e 534f5c3a 534f5c32
001f:00000040  53595332 494e492e 32534f00 4548535f
001f:00000050  433d4c4c 534f5c3a 4d435c32 58452e44
001f:00000060  55410045 54534f54 3d545241 4b534154
001f:00000070  5453494c 4c4f462c 53524544 53455200
```

Figure 22. DoubleWord Format

3.3.13 DA - Display Memory in ASCII Format

Display a range of memory from a given address in ASCII format.

Syntax:

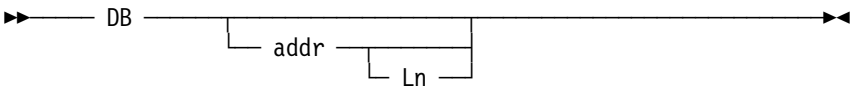


See the D command for a full description.

3.3.14 DB - Display Memory in Byte Format

Display a range of memory from a given address in byte format.

Syntax:

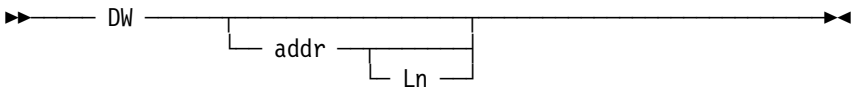


See the D command for a full description.

3.3.15 DW - Display Memory in Word Format

Display a range of memory from a given address in word format.

Syntax:

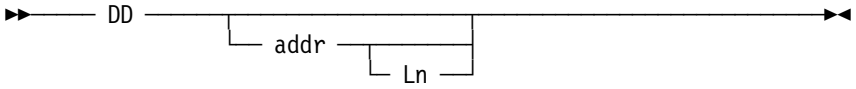


See the D command for a full description.

3.3.16 DD - Display Memory in Doubleword Format

Display a range of memory from a given address in Doubleword format.

Syntax:

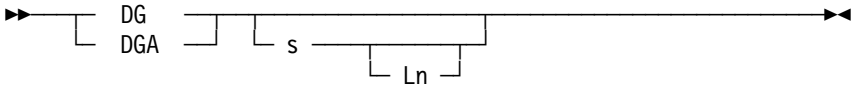


See the D command for a full description.

3.3.17 DG - Display Global Descriptor Table

Display entries from the Global Descriptor Table.

Syntax:



Parameters:

(Default) Display valid GDT entries only.

A Display all GDT entries including invalid descriptors.

s Display descriptor for selector number ***s***.

Notes: Since bit 2 of the selector determines whether the descriptor is local or global the correct table entry will be displayed regardless of whether the DG or DL command is used. If an LDT descriptor is specified then the following message is displayed:

LDT

The requestor priority level (RPL) bits (bits 0 and 1 of the selector) are ignored by DG. Thus: DG 8 displays the same information as DG 9, DG a and DG b.

If the ***s*** parameter is omitted then the entire GDT is displayed.

Ln The number of descriptor entries to display from and including selector ***s***. The default is to display one descriptor entry.

Results and Notes:

One or more descriptor table entries are displayed. An example display follows:

```
##dga
0000  Invalid Bas=00000000 Lim=00000000 DPL=0 NP
0008  Invalid Bas=00000000 Lim=00000000 DPL=0 NP
0010  TSS32   Bas=ffe05dfc Lim=00000067 DPL=0 P  B
0018  Data    Bas=ffe00150 Lim=000003ff DPL=0 P  RW    A  UV
0020  Data    Bas=ffe4a000 Lim=000003ff DPL=0 P  RW    A  UV
0028  LDT     Bas=7ab27000 Lim=0000ffff DPL=0 P
0030  Data    Bas=ffde08a4 Lim=0000575b DPL=0 P  RW ED A  UV
003b  Data    Bas=7c38ba8c Lim=00000073 DPL=3 P  RW
0040  Data    Bas=ffe49400 Lim=000003bf DPL=0 P  RW       UV
004a  Data    Bas=00000000 Lim=1bffffff DPL=2 P  RW    A  G4k BIG UV
0053  Data    Bas=00000000 Lim=1bffffff DPL=3 P  RW    A  G4k BIG UV
005a  Code    Bas=00000000 Lim=1bffffff DPL=2 P  RE C  A  G4k C32 UV
0063  Data    Bas=00000000 Lim=1fffffff DPL=3 P  RW       G4k BIG UV
006b  Data    Bas=00000000 Lim=1bffffff DPL=3 P  RW    A  G4k BIG UV
```

For a detailed explanation of the descriptor table entry format see 3.3.17.1, “Descriptor formats.”

3.3.17.1 Descriptor formats

The Kernel Debugger and Dump Formatter format descriptor table entries in either of two forms depending on whether the descriptor describes a segment of memory or a gate:

```
dddd  type Bas=bbbbbbb Lim=llllllll DPL=p flags

dddd  type Sel:Off=ssss:oooooooo DPL=p flags
```

Each of these fields has the following meaning:

dddd Descriptor number

type Descriptor type. The following are defined:

Table 3 (Page 1 of 2). Descriptor Types

Type	Type Numbers	Description
Code	-	Code segment
Data	-	Data segment
Invalid	0 or 8	Invalid descriptor
TSS	1 or 3	Available or Busy 80286 TSS
LDT	2	system descriptor for an LDT
CallG	4	Call Gate
TaskG	5	Task Gate
IntG	6	80286 Interrupt Gate
TrapG	7	80286 Trap Gate
Reserve	10 or 13	Reserved descriptor types

Table 3 (Page 2 of 2). Descriptor Types		
Type	**Type Numbers**	**Description**
TSS32	9 or 11	Available or Busy Intel486 CPU TSS
CallG32	12	Inter486 CPU Call Gate
IntG32	14	Intel486 CPU Interrupt Gate
TrapG32	15	Intel486 CPU Trap Gate

Bas=bbbbbbbb Segment base address.

Lim=llllllll Segment limit address.

DPL=p Descriptor priority level. Only 0, 2 and 3 are used in OS/2.

Sel=ssss:Off=oooooooo ***selector:offset*** transfer address for a task, interrupt, trap or call gate descriptor.

flags Interpretation of the various descriptor flags. The following abbreviations are used:

Table 4. Descriptor Flags		
Flag	**Bits**	**Description**
NP	¬15	Not present
P	15	Present
RW	9	Read/Write data segment
RO	¬9	Read-only data segment
ED	10	Expand-down data segment
C	10	Conforming code segment
G4k	23	4K granularity segment limit
BIG	22	32-bit offsets used for this data segment
C32	22	32-bit operands used for this code segment
RES	21	reserved
UV	20	Available bit. Used in OS/2 to indicate a UVIRT mapping.
WC=w	0	Word count of a 16-bit call gate
DWC=w	0	Doubleword count of a 32-bit call gate
RE	9	Read/Execute code segment
EO	¬9	Read-only code segment
A	8	Code or Data segment accessed
NB	-	TSS/TSS32 not busy (available)
B	-	TSS/TSS32 busy

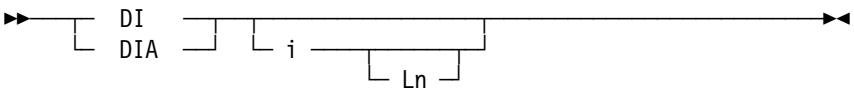
Notes: The bit offsets given above are relative to the second doubleword of the descriptor viewed as 2 doublewords. *The INTEL programmer's Reference* shows the descriptor format as a quad-word, but uses the same offsets specified above.

See the *INTEL Pentium User's Reference* or the *INTEL x86 Programmer's References* for further information.

3.3.18 DI - Display Interrupt Descriptor Table

Display entries from the Interrupt Descriptor Table.

Syntax:



Parameters:

(Default) Display valid IDT entries only.

A Display all IDT entries including invalid descriptors.

i Display descriptor for interrupt vector ***i***.

Ln The number of descriptor entries to display from and including selector ***i***. The default is to display one descriptor entry.

Results and Notes:

One or more descriptor table entries are displayed. An example display follows:

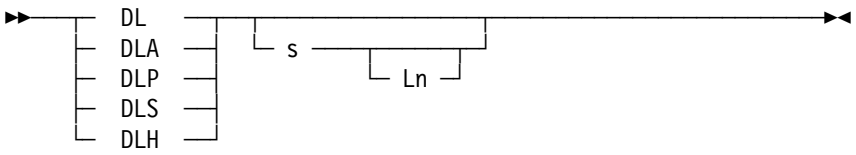
```
##dia
0000  TrapG32 Sel:Off=0170:fff47e64     DPL=0 P
0001  IntG32  Sel:Off=0170:fff47f10     DPL=3 P
0002  TaskG   Sel:Off=1e38:00000000     DPL=0 P
0003  IntG32  Sel:Off=0170:fff480cc     DPL=3 P
0004  TrapG32 Sel:Off=0170:fff48158     DPL=3 P
0005  TrapG32 Sel:Off=0170:fff48164     DPL=0 P
0006  TrapG32 Sel:Off=0170:fff48170     DPL=0 P
0007  TrapG32 Sel:Off=005a:1a090911     DPL=0 P
0008  TaskG   Sel:Off=0088:00000000     DPL=0 P
0009  TrapG32 Sel:Off=0170:fff48258     DPL=0 P
000a  TrapG32 Sel:Off=0170:fff48268     DPL=0 P
000b  TrapG32 Sel:Off=0170:fff48270     DPL=0 P
000c  TrapG32 Sel:Off=0170:fff48278     DPL=0 P
000d  TrapG32 Sel:Off=0170:fff48280     DPL=0 P
000e  TrapG32 Sel:Off=0170:fff4853c     DPL=0 P
000f  TrapG32 Sel:Off=0170:fff48544     DPL=0 P
0010  TrapG32 Sel:Off=0170:fff4854c     DPL=0 P
```

For a detailed explanation of the descriptor table entry format see 3.3.17.1, “Descriptor formats” on page 98.

3.3.19 DL - Display the Current Local Descriptor Table

Display entries from the Local Descriptor Table of the default thread slot. See the **.**S command for information on changing the default thread slot.

Syntax:



Parameters:

(Default) Display valid LDT entries only.

A Display all LDT entries including invalid descriptors.

P Obsolete option. Was used to display only valid private arena LDT descriptors where bits 3 and 4 of the selector number are 0.

S Obsolete option. Was used to display only valid shared arena LDT descriptors where bits 3 and 4 of the selector number are non-zero.

H Obsolete option. Was used to display only huge segment LDT descriptors.

s Display descriptor for selector number ***s***.

Notes: Since bit 2 of the selector determines whether the descriptor is local or global the correct table entry will be displayed regardless of whether the DL or DG command is used. If an GDT descriptor is specified then the following message is displayed:

GDT

The requestor priority level bits (bits 0 and 1 of the selector) are ignored by DL. Thus DL 7 displays the same information as DL 6, DL 5 and DL 4.

If the ***s*** parameter is omitted then the entire LDT is displayed.

Ln The number of descriptor entries to display from and including selector ***s***. The default is to display one descriptor entry.

Results and Notes:

One or more descriptor table entries are displayed. An example display follows:

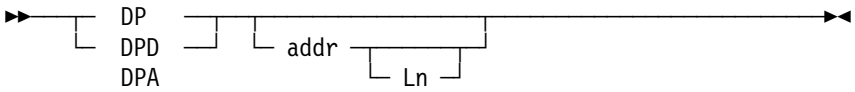
```
##dl
0007  Data    Bas=7ab27000 Lim=0000ffff DPL=3 P  RO
000f  Code    Bas=00010000 Lim=000005ff DPL=3 P  RE
0017  Data    Bas=00020000 Lim=0000005b DPL=3 P  RW
001f  Data    Bas=00030000 Lim=0000fa1f DPL=3 P  RW    A
0027  Data    Bas=00040000 Lim=00000276 DPL=3 P  RW    A
002f  Data    Bas=00050000 Lim=00000fff DPL=3 P  RW
0036  Data    Bas=00060000 Lim=00003fff DPL=2 P  RW    A
003f  Data    Bas=00070000 Lim=00000fff DPL=3 P  RW    A
0047  Data    Bas=00080000 Lim=00000fff DPL=3 P  RW
004f  Data    Bas=00090000 Lim=0000ffff DPL=3 P  RW    A
0056  Code    Bas=000a0000 Lim=00000af7 DPL=2 P  RE C
005f  Data    Bas=000b0000 Lim=0000ffff DPL=3 P  RW
```

For a detailed explanation of the descriptor table entry format see 3.3.17.1, “Descriptor formats” on page 98.

3.3.20 DP - Display Page Directory and Table Entries

Display entries from the page tables of the default thread slot. See the **:S** command for information on changing the default thread slot.

Syntax:



Parameters:

A Display both page table and page directory entries. This is the default.

D Display only page directory entries.

addr The linear or virtual address whose page directory and table entries are to be displayed. If not specified DP displays the entire page directory and its page tables.

An address expression may be specified.

L*n* The number of page table entries to display starting with the entry for ***addr***. The default is to display the all page table entries from this entry assigned to ***addr***.

> **Note**
>
> Due to a bug in some versions of the Kernel Debugger an extra zero is required for this parameter.

Results and Notes:

One or more page and directory table entries are displayed. An example display follows:

```
DP %90000 L50

 linaddr   frame   pteframe  state res Dc Au CD WT Us rW Pn state
%00090000* 012f3  frame=012f3  2    0  D  A        U  W  P  resident
%00090000         vp id=00a76  0    0  c  u        U  W  n  pageable
%000a0000  000b8  vp id=000b8  1    0  D  u        U  W  n  uvirt
%000b0000  00888  frame=00888  0    0  D  A        U  W  P  pageable
%000c0000         vp id=00b8f  0    0  c  u        U  W  n  pageable
%000d0000         vp id=00b92  0    0  c  u        U  W  n  pageable
##
```

Output from the DP command is presented in tabular form. Each of the columns shown is described as follows:

linaddr Linear address of virtual memory whose page directory and table entries are being formatted. Those lines corresponding to directory entries have an ***** flag suffixed to the linear address. Page table entries for a given directory entry are formatted following the directory entry.

In the example above the linear address **%90000** has its page table located in physical frame **12f3**, that is at physical **%%12f3000**. The page table entry corresponding to virtual memory at **%90000** is described in the second line. Each of the following lines are consecutive entries from page table **12f3**.

frame The real storage frame number that contains either the page table (***** suffix to **linaddr**) or page frame corresponding to the **linaddr**. If this field is blank then the frame has been discarded. If it contains a frame number then the contents are still valid even though the page table entry no longer points to a page frame. See **pteframe** field for further discussion.

pteframe For table entries with the present bit set the this field shows the page frame number pointed to by this table entry. This is shown as **frame=*fffff***. Use the frame number with the **.**MP command to obtain information on allocation and ownership of this this frame of real storage.

For decommitted pages the table entry contains the Virtual Page ID. This is shown as **vp id=*vvvvv***. Use the **.**MV command with the virtual page Id to obtain information on allocation and ownership of this memory.

Notes: The ***vp id*** is not valid to use with **.MV** if the **state** of the table entry is uvirt.

If the frame has been decommitted but the **frame** field still shows a frame number then the frame contents are still valid for reclaiming without a page-in operation from the swap file. The corresponding virtual page will be queued from the idle list. See **.**MV and **.**MP commands for more information on page management.

state State information is stored in the **available** bits (9 - 11) of the page table entry. These are interpreted on the right-hand end of the display. The following values may appear:

State	Value	Description
pageable	0	Storage may be paged-out to the swap file
uvirt	1	Physical to virtual mapping reservation only.
resident	2	Non-pageable fixed storage
uvirt	3	Physical to virtual mapping reservation only.

Res Reserved page table entry bits. Should always be zero

Dc Set to **D** if the page is dirty, otherwise **c** (clean).

Au Set to **A** if the page has been accessed, otherwise **u** (unaccessed).

CD Set to **CD** if the TLB cache-disable bit is set, else blank.

WT Set to **WT** if the TLB cache write-transparent bit is set, else blank.

Us Set to **U** if the page is for user storage, otherwise **s** (supervisor).

rW Set to **r** if the page is read-only, otherwise **W** (writeable).

Pn Set to **P** if the page is present, otherwise **n** (not present).

Refer to the following for more information on page and memory management:

.M family of Kernel Debugger and Dump Formatter commands

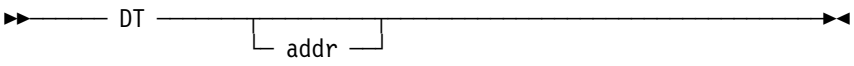
Intel Pentium User's Guide

Intel x86 Programmer's Reference

3.3.21 DT - Display a Task State Segment

Format a task state segment.

Syntax:



Parameters:

addr The address of the task state segment to be formatted. If not specified then the current TSS pointed to by the **TR** (task register) is used.

Results and Notes:

The TSS is formatted as follows:

```
##dt 10:0
eax=00000000 ebx=00000000 ecx=00000000 edx=00000000 esi=00000000 edi=00000000
eip=00000000 esp=00000000 ebp=00000000 iopl=0 -- -- -- nv up di pl nz na po nc
cs=0000 ss=0000 ds=0000 es=0000 fs=0000 gs=0000  cr3=001d1000
ss0=0030  esp0=000066fc  ss1=0000  esp1=00000000  ss2=0866  esp2=00001000
ldtr=0028  link=0000  tflags=0000  i/o map=dfff
ports trapped: 0-ffff
```

Each of the fields displayed has the following meaning:

Note: Some of the **TSS** fields are set at task creation and other when a task switch occurs.

eax= Saved **EAX** register when a task switch occurs.

ebx= Saved **EBX** register when a task switch occurs.

ecx= Saved **ECX** register when a task switch occurs.

edx= Saved **EDX** register when a task switch occurs.

esi= Saved **ESI** register when a task switch occurs.

edi= Saved **EDI** register when a task switch occurs.

eip= Saved **EIP** register when a task switch occurs.

esp= Saved **ESP** register when a task switch occurs.

ebp= Saved **EBP** register when a task switch occurs.

iopl= Saved **EGLAGS** iopl and flag settings when a task switch occurs. See the .R command for an explanation of the flag abbreviations.

cs= Saved **CS** register when a task switch occurs.

ss= Saved **SS** register when a task switch occurs.

ds= Saved **DS** register when a task switch occurs.

es= Saved **ES** register when a task switch occurs.

fs= Saved **FS** register when a task switch occurs.

gs= Saved **GS** register when a task switch occurs.

cr3= **CR3** register at task creation.

Note: This provides the real address of the Page Directory Table, which never alters under OS/2.

ss0= Ring 0 **SS** register used for ring 0 privilege transitions.

esp0= Ring 0 **ESP** register used for ring 0 privilege transitions.

ss1= Ring 1 **SS** register used for ring 1 privilege transitions.

esp1= Ring 1 **ESP** register used for ring 1 privilege transitions.

Note: Ring 1 is not used under OS/2.

ss2= Ring 2 **SS** register used for ring 2 privilege transitions.

esp2= Ring 1 **ESP** register used for ring 2 privilege transitions.

ldtr= **LDTR** register at task creation.

link= **TR** register value of previous nested task's **TSS.**

tflags= The debug trap bit for this task.

i/o maps= Offset to the I/O permission map from the beginning of this TSS.

Note: It is permissible for the I/O map offset to point beyond the TSS segment. This signifies that no I/O permissions are granted and all ports will be trapped.

ports trapped: List the range of I/O port addresses that will generate traps if accessed by this task.

Notes: For performance reasons hardware implemented task switching is used only in a limited way in OS/2. **TSSs** defined by OS/2 include:

Protect mode code (**TSS** selector **10**)

Virtual DOS Machines

Non-Maskable Interrupt handling (trap 2, **TSS** selector **1E38**)

Double Fault handling (trap 8, **TSS** selector **88**)

All protect-mode processes run under a common top-level task using selector 10 as the **TSS** selector.

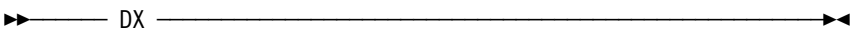
The **seltss (PTDA +0x2f0)** field of the PTDA records the top-level task's **TSS** selector used by a given process; thus it may be used to find the **TSS** selector for Virtual DOS Machines.

Refer to *Intel Pentium User's Guide* and *Intel x86 Programmer's Reference* for more information on the Task State Segment and Hardware architected multitasking.

3.3.22 DX - Display the 286 LoadAll Buffer

Formats the 286 LoadAll buffer from physical address **%%800** in memory.

Syntax:



Parameters:

None.

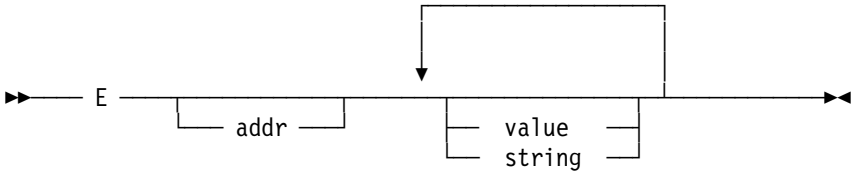
Results and Notes

This command applies to the Intel 286 processor and is now obsolete. The results are meaningless.

3.3.23 E - Enter Data into Memory

Enter data into a memory location.

Syntax:



Parameters:

addr The address of the memory location to be changed. If not specified this defaults to **DS:00000000** where **DS** is established by the most recent register display.

An address expression may be specified. See the R and .R commands for information on establishing default addresses.

value A numerical byte value to be entered into memory. One or more values may be specified separated commas or blanks. These may be mixed with *"**string**"* values.

string A character sting enclosed in quotes. Each character is treated as a byte value and entered into memory separately, no terminating 0x00 value is stored. No folding of characters to upper or lower case occurs. One or more strings may be specified separated by commas or blanks. These may be mixed with numerical ***values***.

Results and Notes:

If memory is present values are entered into storage otherwise an **Invalid Address** message is generated. If this should happen, valid storage may be paged into memory by means of the **.**I command.

If no ***value*** or ***string*** parameter is specified the Kernel Debugger prompts the user a byte at a time for replacement values by displaying the original value followed by a colon. In prompt mode, the user may proceed as follows:

Type a replacement byte value in hexadecimal, or

Accept the original value and move on to the next location by pressing the space-bar, or

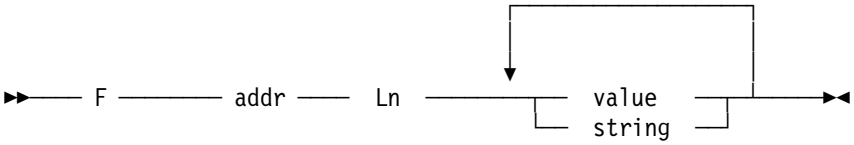
Back up to the previous location by entering a **-** (minus) character, or

Terminate prompt mode by pressing carriage return (with or without a replacement value).

3.3.24 F - Fill Memory with Repeated Data

Fill memory with repeated data.

Syntax:



Parameters:

addr The address of the memory location to be changed.

An address expression may be specified.

Ln The number (***n***) of bytes to fill with data.

value A numerical byte value to be entered into memory. One or more values may be specified separated commas or blanks. These may be mixed with *"**string**"* values.

string A character sting enclosed in quotes. Each character is treated as a byte value and entered into memory separately, no terminating 0x00 value is stored. No folding of characters to upper or lower case occurs. One or more strings may be specified separated by commas or blanks. These may be mixed with numerical ***values***.

Results and Notes:

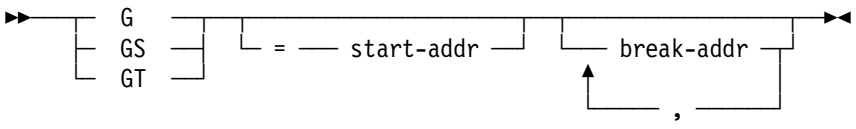
The list of ***values*** and ***strings*** is repeated up to the length **L*n*** and used to fill memory at the specified address. If the fill data is shorter than the length then it is repeated; if it is longer, it is truncated.

If memory is present the storage is updated. Otherwise an **Invalid Address** message is generated. If this should happen, valid storage may be paged into memory by means of the **.**I command.

3.3.25 G - GO

Cause execution to continue from a given point and optionally set 1 or more go breakpoints.

Syntax:



Parameters:

(Default) Continue execution from the current **CS:EIP**.

S The *go-special* command causes the high-resolution time interval to be recorded from the point GS command is issued to the point that the Kernel Debugger is re-entered as the result of a breakpoint firing.

Notes: No account is taken of the Kernel Debugger overhead when calculating the time interval.

When the Kernel Debugger re-enters, for whatever reason, the interval timer is cancelled until another GS command is executed.

If the reason for entry is for reasons other than the firing of a sticky or go breakpoint then in addition to cancelling the interval timer no time message displayed.

T This option causes the Kernel Debugger's trap vector handlers to be removed temporarily from the IDT and the system's re-instated until after then next instruction has executed. After execution of the next instruction the the Kernel Debugger's V commands are re-instated.

This is a convenience option that saves manually unhooking a Kernel Debugger trap vector handlers from the IDT, using a command sequence similar to:

```
VC n
T
VS n
G
```

start-addr The address from which execution is to continue. This must be a valid address for the current context. If ***start-addr*** is omitted then execution continues from the current CS:EIP, as shown by the R command.

Information

Be very careful to ensure that the start address is valid for the privileged level and addressability of the code and data selectors in use. If the Kernel Debugger attempts to load a segment register that is invalid, the system may trap in the debugger code.

break-addr Up to ten go breakpoints may be specified. These are temporary breakpoints set in addition to any sticky breakpoints set by the B commands. When the Kernel Debugger is next entered, for whatever reason, all go breakpoints are cleared.

If ***break-addr*** is omitted then the system continues execution until:

A fatal exception occurs

An Internal Processing Error (IPE) occurs.

A sticky breakpoint fires

A non-maskable interrupt occurs

An INT 3 instruction is executed

The user enters Ctrl-C from the debugging console

Results and Notes:

The system continues execution until the Kernel Debugger is re-entered. If the reason for entry is other than a breakpoint firing then the R command is automatically executed followed by one of the following command prompts:

> (signifies a command prompt in real mode)

(signifies a command prompt in protect mode with paging disabled)

- (signifies a command prompt in V86 mode with paging disabled)

(signifies a command prompt in protect mode with paging enabled)

-- (signifies a command prompt in V86 mode with paging enabled)

If an error situation caused entry to the Kernel Debugger then a diagnostic message may be generated by the failing code writing directly, to the Kernel Debugger's communications port.

If entry was caused by a Kernel Debugger trap handler receiving control then a message from the trap handler will be displayed. See the V command for details.

If a breakpoint caused the Kernel Debugger to receive control then commands associated with the breakpoint that fired will execute. See the B commands for details.

If a go-special was interrupted by a breakpoint firing then the following message appears before any output associated with the breakpoint:

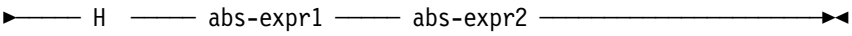
```
Go Time (tics) = 017fb (hex) = 5145 (uSec)
```

This shows the time interval in both timer-ticks and equivalent number of micro-seconds.

3.3.26 H - Hex Arithmetic

Display the sum, difference, product, quotient and remainder of two absolute expressions.

Syntax:



Parameters:

abs-expr1 An expression that resolves to a simple numeric value using any of the expression evaluator operators. Symbols of absolute values may be specified in the expression, but symbols of relocatable addresses may not.

abs-expr2 An expression that resolves to a simple numeric value using any of the expression evaluator operators. Symbols of absolute values may be specified in the expression, but symbols of relocatable addresses may not.

Results and Notes:

Each of the expressions is evaluated. If either does not resolve to a simple numeric value then the following message is displayed:

```
Expression error
```

Having resolved each of the expressions then the sum, difference, product and quotient of the pair is displayed as in the following examples:

```
##h 2 3
+0005  -ffff  *0006 0000  /0000 0002
#h 10t 5
+000f  -0005  *0032 0000  /0002 0000
##h 7fff 5
+8004  -7ffa  *7ffb 0002  /1999 0002
## 5*4 2*3
+001a  -000e  *0078 0000  /0003 0002
##h bmp_segsize 5
+0017  -000d  *005a 0000  /0003 0003
##h
```

Notes: Calculations are performed using 16-bit signed arithmetic.

The operation performed is shown prefixing the result.

The product is shown as a two word value, the low word followed by the high word.

The division is shown as two words, the quotient followed by the remainder.

In the last example, *bmp_segsize* is an absolute symbol of value *0x0012* defined in map OS2KRNL.

Evaluation of complex expressions involving relocatable addresses may be done using the ? command.

Information

A simple numeric expression is one that resolves to a single integer value, for example:

```
-4
55c7
```

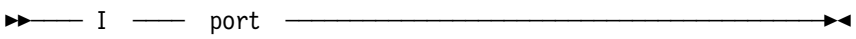
Compare this with an address expression that has in addition an address operator (&, %, %%, #) and possibly involves more than one integer value, for example:

```
&1fc:45
#1f:445
%30045
%%15c
```

3.3.27 I - Input from an I/O Port

Input a byte of data from a 16-bit I/O port.

Syntax:



Parameters:

port A 16-bit I/O port address. This may be specified as a simple numeric expression.

Results and Notes:

The byte of data is read from the requested I/O port and displayed in hexadecimal at the console. For example:

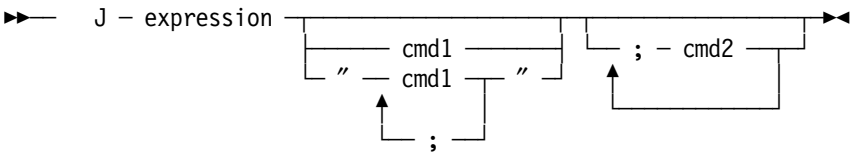
```
##I 2f8
0d
```

See 3.3.32, “O - Output to an I/O Port” on page 123 for related information.

3.3.28 J - Execute Commands Conditionally

Conditionally execute one of two lists of commands depending on whether an expression evaluates to TRUE (non-zero) or FALSE (zero).

Syntax:



Parameters:

expression An expression that resolves to either a simple numeric value or an address using any of the expression evaluation operators. Symbols of addresses and symbols of absolute values may be specified.

cmd1 Specifies a command to be executed if the ***expression*** evaluates to TRUE (non-zero). More than one command may be specified if each is separated by a semi-colon and the entire command list is enclosed in single or double quotes.

If ***cmd1*** is omitted, control is returned to the debugging console when the ***expression*** is TRUE.

cmd2 Specifies a command to be executed if the ***expression*** evaluates to FALSE (non-zero). More than one command may be specified. Each ***cmd2*** must be prefixed by a semicolon, even if only one is specified. Quotes are not required to encompass a list of

If ***cmd2*** is omitted, control is returned to the debugging console when the ***expression*** is FALSE.

Results and Notes:

If the expression resolves to one of the following forms, it is considered to be FALSE:

```
0
0:0
&0:0
%0
%%0
```

Any other resolution is regarded as TRUE.

The J command is primarily intended to be used with the BP and BR commands to enable conditional breakpoints to be defined.

Examples of this usage are:

```
BP #f:12d5 "J ax!=10t,g"

BP #f:12d5 "J ax==10t;g"

BP SchedNextRet "J wo(Tasknumber)==8,'.p*;.r';g"

BP DOSOPEN "J wo(Tasknumber)==32,'da #(wo(ss:sp+20)):(wo(ss:sp+1e));g';g"
```

The first example shows a breakpoint set at address **#f:12d5**. When this breakpoint fires the J command tests the condition of the AX register not equal to decimal 10. If this is true, the G command is executed. Since no ***cmd2*** is specified the J command returns control to the debugging console when the condition is FALSE (AX equal to decimal 10).

The second example is has the same effect as the first but is implemented by testing the logically opposite condition.

The third example shows one method of stopping the system when a thread switch to a particular thread slot has just occurred. In this case the debugging console gains control when thread slot 8 is selected, whereupon **.**P* and **.**R commands are automatically executed. The breakpoint *SchedNextRet* is one of two exit points from the scheduler (*_tkSchedNext*). The other, *SchedNextRet2* is taken when the same thread slot is selected for re-dispatch. The global variable *Tasknumber* contains the current and therefore out-going slot number on entry to the scheduler; and in-coming slot number on exit from the dispatcher.

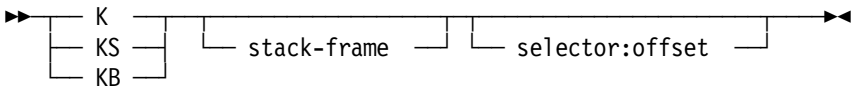
Note: The kernel calls one of the *KMExitKmode* routines before giving control to user code. During this kernel exit processing the *Resched* and (TCB and PTDA) force flags are checked again and if set the scheduler/dispatcher sequence is invoked. It is possible therefore, that even though a thread is selected to run, and achieves run state, it is put back on the ready queue before being given any user processing time.

The fourth example illustrates a method of tracing resources that are opened by a specific thread slot (in this case slot 32) without giving control to the debugging console. DOSOPEN is the kernel's entry point for open processing. At this point words **0x0f** and **0x10** contain the offset and selector that points to the resource name.

3.3.29 K - Display Stack Trace from Address

Display the stack-trace from a given stack frame address.

Syntax:



Parameters:

K Display stack frame trace assuming the default operation size from the code descriptor specified by ***selector:offset***

KS Display frame trace assuming an operation size of 16-bits (small-model).

KB Display frame trace assuming an operation size of 32-bits (big-model).

stack-frame Address of the starting stack-frame. If not specified then this defaults to the current **SS:EBP** or **SS:BP** as set by the last register display.

See the R and .R commands for information on changing the default register values.

An address expression may be specified.

selector:offset The ***selector:offset*** address of the code that is in effect when the starting ***Stack-frame*** address was created. If not specified this defaults to the current **CS:EIP** or **CS:IP** as displayed by the the R command.

The code selector associated with this address is used for two purposes:

1. To determine the default operand size in effect from the code segment descriptor.
2. To attempt to distinguish between near and far calls at the starting ***stack-frame*** address.

Results and Notes:

The K command displays the stack trace, threading through the **BP** or **EBP** chain until either an invalid chain pointer is encountered or the command is interrupted by the user. For each stack-frame, the return address and for parameter words or doublewords are displayed. The symbol associated with the return address is displayed after the parameter words. An example is given below:

```
##.S 8
##.R
eax=c7c00000 ebx=00000014 ecx=003acd7 edx=0000aff7 esi=00030bff edi=00030000
eip=0000272d esp=0003f8b8 ebp=0003f8c0 iopl=2 -- -- -- nv up ei ng nz na pe nc
cs=d02f ss=001f ds=aff7 es=be47 fs=150b gs=0000  cr2=1701d000  cr3=001d9000
doscall1:CONFORM16:postDOSSEMWAIT:
d02f:0000272d c9              leave
##K SS:BP CS:IP

bdef:0000711e ffff ffff 06d6 0a23 SEEPSMQ + 67
bdef:0000e1df ffff ffff 0bff f91c GETMSGNOINPUT + 4a
be1f:00000271 8001 ffff 0000 0000 THK16_CALLUSERTHUNKPROC + 12
be1f:00000003 05ae 0001 0001 0003 THUNKTOINITMOVECURSOR + 30020:00000003 0001 0001
0020:00000000 02af 1a03 0197 0000
##
```

Notes:

1. The K command is insensitive to the unconventional use of the stack, such as where subroutine returns are affected explicitly by setting the stack pointer and jumping back to the calling code or in optimized code where the **EBP** or **BP** registers are not used as stack-frame pointers.

   Such possibilities exist within the system when for example the kernel returns to user code and also within some Presentation Manager components.

2. No attempt is made to trace correctly through thunking layers where the default operand size changes.

3. The stack trace is insensitive to any explicit segment operand overrides that may be active.

4. No attempt is made to examine the descriptor of the **SS** register to determine whether **EBP** or **BP** should be used. In a lot of 32-bit code both the 16-bit and 32-bit data descriptors are created by the system for calls to 16-bit subroutines.

   In the example above the stack-frame address has been explicitly overridden to use **BP** since the 16-bit stack selector (**1f**) is in effect rather than the 32-bit **53** selector.

5. Unlike the default stack-frame address the default code ***selector:offset*** is taken from the register values on entry to the Kernel Debugger.

Attention

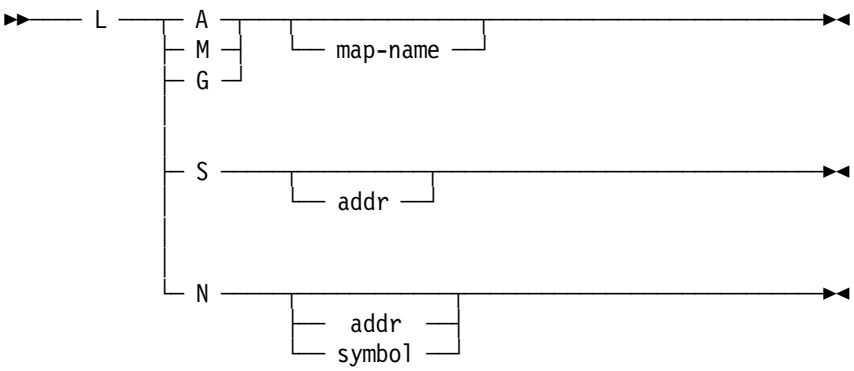
In consequence of these points, it recommended that the ***stack-frame*** and code ***selector:offset*** addresses be explicitly coded when using the K command, as in the example above. In addition, the stack trace should be verified with a memory dump of the stack.

3.3.30 L - List Maps, Groups and Symbols

List maps, groups and symbols from loaded symbol files.

See the W command for related information.

Syntax:



Parameters:

A List absolute symbol definitions for the specified ***map-name*** or for all active maps.

M List all active maps or the status of the specified map.

G List groups defined in all active maps of the specified map.

N When ***addr*** is specified this option lists the nearest symbols to the address. If an exact match is found, symbols are listed; otherwise, the nearest symbol before and after ***addr*** is listed.

When ***symbol*** is specified the address, map and group corresponding to the symbol is listed.

If neither ***addr*** nor ***symbol*** is specified then the default disassembly address is assumed. See the **.**R and U commands for related information.

S List all symbols defined in the group that encompasses ***addr*** for all active maps. If ***addr*** is not specified then the value of **CS:EIP** on entry to the debugger is assumed, as displayed by the R command.

map-name Specifies the link edit map-name from which information is to be displayed.

addr Specifies an explicit address expression.

symbol Specifies a publicly defined symbol name from a program source code.

Results and Notes:

Symbol maps are obtained from symbol files (*.SYM), which are generated using the linkage editor and the MAPSYM utility. Under the Kernel Debugger they are loaded from the same directory as their corresponding load module when that is

loaded by the system. When this happens the **Symbols linked (*map-name*)** message appears. When a load module is deleted from the system, its map is removed and the message **Symbols unlinked (*map-name*)** appears.

Under the Dump Formatter symbol files are loaded for each MTE in the dump, during initialization, from the current directory (usually the directory the Dump Formatter is running from).

Under the Dump Formatter conforming segments are not checked. Thus a ring 2 selector:offset address may not be recognized, whereas the ring 3 selector is. If LN does not find a symbol for a ring 2 selector, try specifying the same selector with the ring 3 RPL specified. For example, specify **d0fe:1234** as **d0ff:1234**.

Under the Dump Formatter LN does not check equivalences of the selector:offset and linear forms of an address. Therefore it may be necessary to apply the CRMA to an address if the LN command fails to find any near symbols.

Loaded symbol maps may be active or inactive, depending on whether the corresponding load module is (potentially) active in the current context. In the case of private executable modules erroneous symbolic information may be associated with a private storage location. For this reason maps may be manually activated and removed using the W command.

Maps for WIN-OS2 and Windows components are supported under the Kernel Debugger only. These are automatically activated and deactivated according to whether the Kernel Debugger default thread slot is a Windows or WIN-OS2 environment.

Output from each of the L subcommands is more or less self explanatory. Examples follow:

```
##la
cmd:
9876 __acrtmsg
9876 __acrtused
d6d6 __aDBused
d6d6 __aDBdoswp
```

Figure 23. List Absolute Symbols Defined in CMD.EXE and their Associated Constants

```
##lm
cmd is active
kernel [0040, 003f]
minxobj is active
wpprint is active
nwiapi is active
rexxinit is active
pmmle is active
fka is active
ibmdevr is active
ibmvgar is active
pmpre is active
os2krnl is active
```

Figure 24. List Current MAP Status

Note: The Windows Kernel is not active, but loaded in thread slots **40** and **3f**. The additional active slot number information is only provided with Windows and WIN-OS2 environment map files.

```
##lg cmd
cmd:
000f:00000000 _TEXT1
0017:00000000 _TEXT3
001f:00000000 DGROUP
```

Figure 25. List Segment Groups Defined in CMD.EXE and their Associated Addresses

```
##ln %20000
%00020000 cmd:_TEXT3:_eChcp
##ln _tkschednext
%fff4521c os2krnl:DOSHIGH32CODE:_tkSchedNext
##ln
0170:fff44695 os2krnl:DOSHIGH32CODE:HaltInst + 1
0170:fff44787 postSchedNext - f1
```

Figure 26. List Near Symbols and their Associated Addresses

Note: In this example three uses of LN are shown:.

1. Address **%20000** is shown to coincide with **_eChcp** in the **_TEXT3** group of CMD.EXE.
2. Symbol **_tkschednext** is shown to be at address **%fff4521c** in the **DOSHIGH32CODE** of **OS2KRNL**.
3. The current **CS:EIP** is at **+ 1** byte from **HaltInst** in group **DOSHIGH32CODE** of module **OS2KRNL** and **-f1** bytes before **postSchedNext** in the same group and module.

```
##ls %fff3f500
%fff3f4a4 DevWOHandle
%fff3f4ac g_CodeLockProc
%fff3f4b1 CodeLockProc
%fff3f5a4 g_CodeUnlockProc
%fff3f5a9 CodeUnlockProc
%fff3f614 _FSAbortVDM
%fff3f62c FS32IREAD
%fff3f638 FS32IWRITE
%fff3f644 w_Big32IO
%fff3f6c0 w_SetFileLocks
%fff3f6c8 w_ProtectSetFileLocks
.
.
.
```

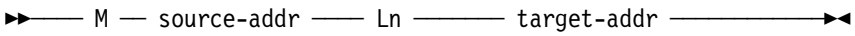
Figure 27. List Symbols in the Current Group Encompassing Address %fff3f500

See the W command for related information.

3.3.31 M - Move a Block of Data in Memory

Move a block of contiguous data from one memory location to another. This command guarantees to duplicate the source data even when the source and destination overlap.

Syntax:



Parameters:

source-addr The source address of the memory location to be moved (copied).

An address expression may be specified.

Ln The number (n) of bytes to move.

target-addr Target address of the memory move operation.

An address expression may be specified.

Results and Notes:

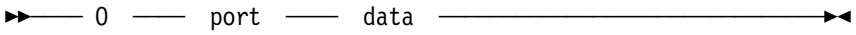
Memory content is copied from the source to the target address. If the source and target overlap then the source will be updated; however, the move operation is conducted from highest to lowest address or *vice versa* depending on whether the target address is higher or lower than the source, thereby guaranteeing a faithful copy of the original source.

If memory is present, the storage is updated; otherwise an *Invalid Address* message is generated. If this should happen, valid storage may be paged into memory by means of the `.I` command.

3.3.32 O - Output to an I/O Port

Output a byte of data to a 16-bit I/O port

Syntax:



Parameters:

port 16-bit I/O port address.

data A byte of data expressed numerically. This may be specified as a simple numeric expression.

Results and Notes:

The byte is sent to the requested I/O port.

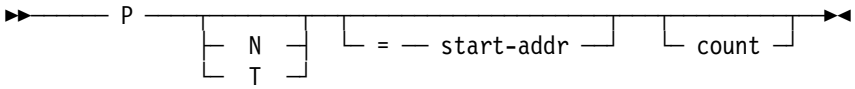
Note: This command may be used to set the debugging communication port parameters from the Kernel Debugger initialization command file (KDB.INI) as in the following example:

```
Set COM2 DTR line (assume standard port assignment for COM2 that is, 2f8):
##O 2fc 1

Set COM1 DTR line (assume standard port assignment for COM1 that is, 3f8):
##O 3fc 1
```

3.3.33 P - PTrace Instruction Execution

Trace instruction execution within a single procedure. This command is very similar to the T command, except that the CALL, loop and string repeat instructions are traced as single instructions (even though allowed to execute correctly).

Syntax:



Parameters:

(Default) Trace instruction execution by single-stepping, treating CALL, loop and string repeat instructions as single events.

Note: Certain areas of the system are known to cause problems if traced. Attempts to trace these areas are intercepted by the Kernel Debugger. See below for further information.

N Trace instructions suppress the register display after each instruction is executed.

T This option causes the Kernel Debugger's trap vector handlers to be removed temporarily from the IDT and the system's re-instated until after the next instruction has executed. After execution of the next instruction the Kernel Debugger's V commands are re-instated.

This is a convenience option that saves manually unhooking a Kernel Debugger trap vector handlers from the IDT using a command sequence similar to:

```
VC n
P
VS n
```

start-addr The address from which the execution is to continue. This must be a valid address for the current context. If ***start-addr*** is omitted, then execution continues from the current **CS:EIP**, as shown by the R command.

count The number of instructions to trace before re-entering the Kernel Debugger, unless one of the following conditions is encountered:

A fatal exception occurs.

An Internal Processing Error (IPE) occurs.

A sticky breakpoint fires.

A non-maskable interrupt occurs.

An INT 3 instruction is executed.

The user enters Ctrl-C from the debugging console.

If omitted then ***count*** defaults to one instruction.

Results and Notes:

The **Ptrace** commands trace the execution of machine instructions, and by default, display the current registers and next instruction to execute at each step. For the purposes of the displayed trace, the **CALL** instruction does not have the called routine traced, but tracing resumes on return. Loop and string repeat instructions are also treated as atomic entities with the instruction following the loop or repeat shown as the next to execute. INT 3 instructions are stepped over to avoid a double breakpoint at the same address even though they appear as the next instruction to execute.

The following system routines are known to causes inconsistency or even system failure if traced. Consequently **Ptrace** will suspend tracing until after execution leaves these routines.

```
_Debug_CtrlC32   through _EndCtrlC32
_DebugLoadSymMTE through EndDebugLoadSymMTE
_PGSwitchContext through pgSwitchRet
```

See the TX command for information on tracing these routines.

PN suppresses the register display from the automatic R command, but still displays an unassembled next instruction for each traced instruction. If the ZS command has been used to specify a different default command then PN behaves exactly as the P command.

An example of the output from PN is as follows:

```
##PN 5
0170:fff4521f 803d9e53e0ffff cmp     byte ptr [InterruptLevel (ffe0539e)],ff
0170:fff45226 75b4           jnz     fff451dc
0170:fff45228 803d9643e0ff00 cmp     byte ptr [_cTKNoBlock (ffe04396)],00
0170:fff4522f 75be           jnz     fff451ef
0170:fff45231 0f01e1         smsw    cx
##
```

Note: The last traced instruction is the next to be executed.

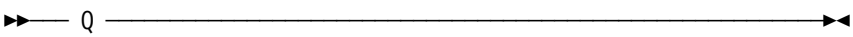
Attention

If any of the **PTrace** commands is interrupted, the Kernel Debugger may leave a temporary breakpoint active. This will result in a *Trap 1* when the system is next given control. If this occurs then either of the PT or GT commands will clear this condition.

3.3.34 Q - Quit the Dump Formatter

Quit the Dump Formatter.

Syntax:



Parameters:

None

Results and Notes:

The Dump Formatter is terminated.

Attention

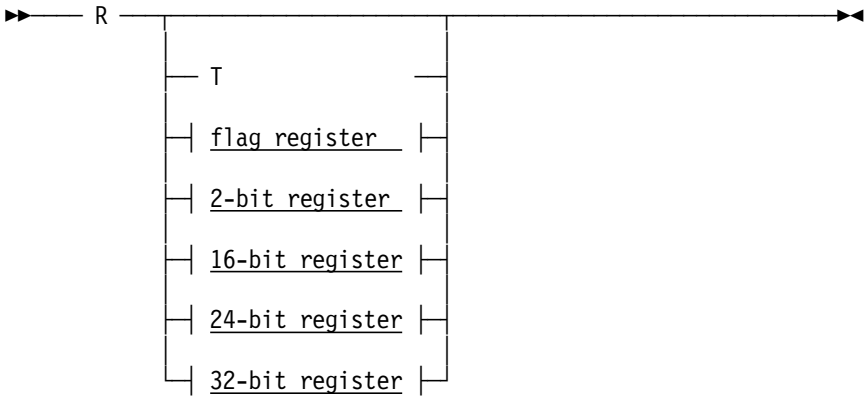
Do not use this command when the Dump Formatter is invoked from PMDF. This will cases PMDF to hang. To terminate the Dump Formatter either quit PMDF from the system memu or select another dump for processing.

3.3.35 R - Set or Display Current CPU Registers

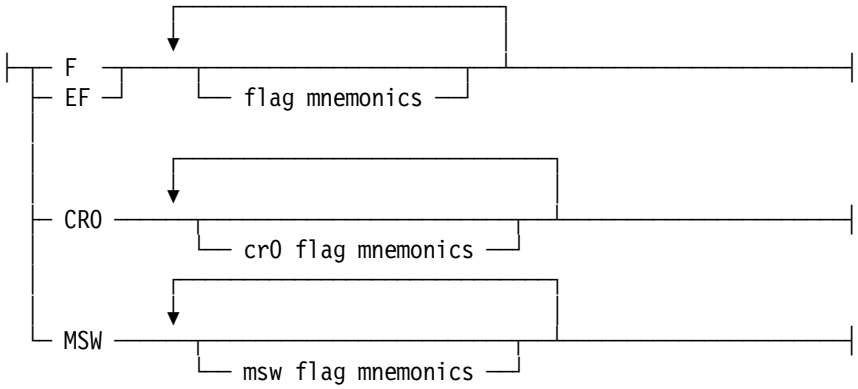
Display or set the current CPU registers saved on entry to the Kernel Debugger. Set default addresses for the E, D, K and U commands.

Under the Dump Formatter this command is implemented as an alias to the .R command. The remaining discussion in the section applies to the Kernel Debugger.

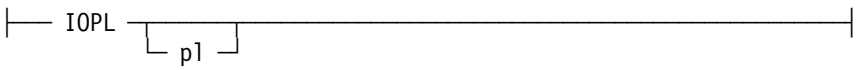
Syntax:



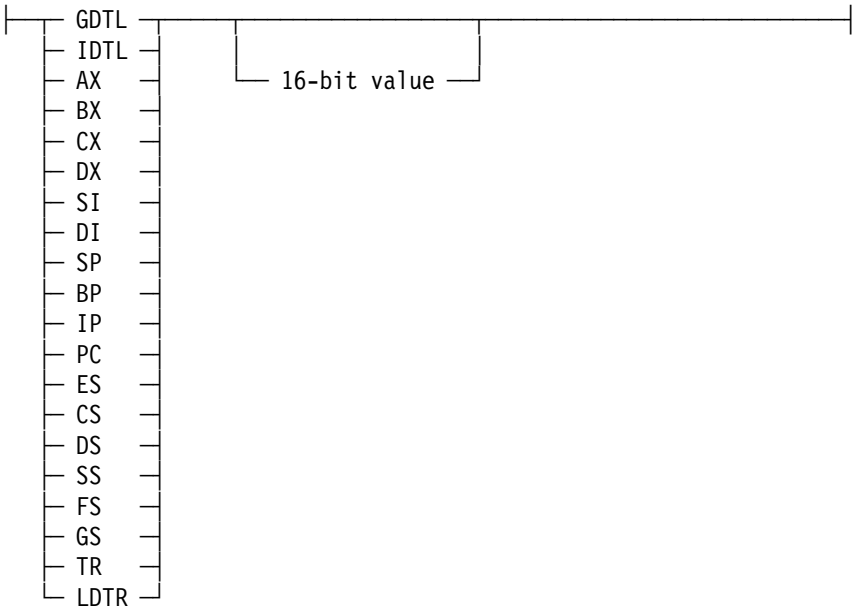
flag register:



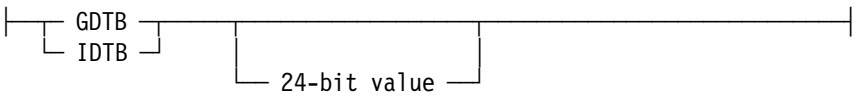
2-bit register:



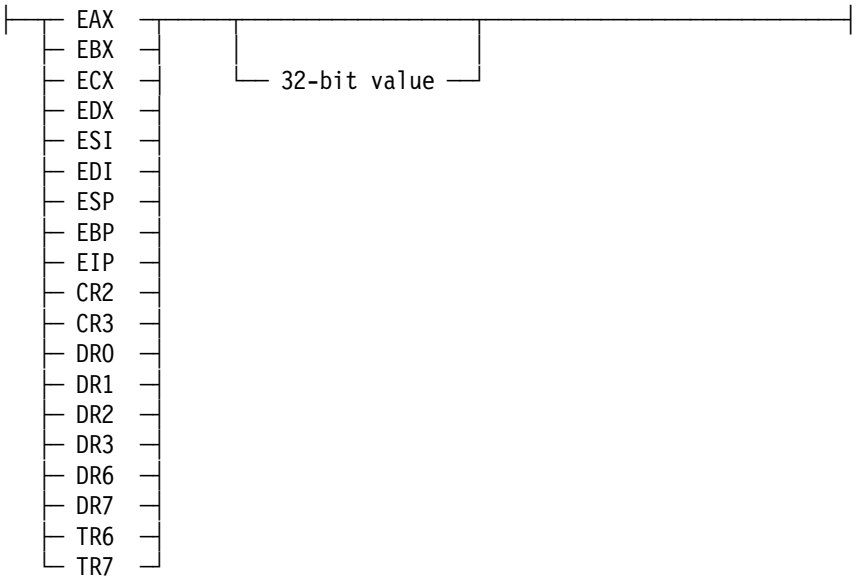
16-bit register:



24-bit register:



32-bit register:



Parameters:

(Default)
Displays the current CPU registers on entry to the Kernel Debugger and sets default addresses for E, D and U commands.

Register mnemonics are assigned the values displayed for use in address expressions and operands of other Kernel Debugger and Dump Formatter commands.

Note: The **.**SS command may be used to change the displayed values of CS, EIP, SS and ESP. It does not affect the values restored then the Kernel Debugger returns control to the system.

T Toggle register display mode between terse and non-terse forms. The terse form suppresses display of the test, debug, control, descriptor table and task registers.

This option affects both the R and **.**R commands.

flag register

Specifies one of the flag registers to be modified. The following mnemonics may be used:

F 80286 **FLAGS** register

EF 80486 **EFLAGS** register

MSW Machine status word

CR0 Control register 0

Each of the flag bits is specified by a mnemonic. More than one flag may be specified, with the order being unimportant. The Kernel Debugger processes the flags from left to right; if an invalid flag is encountered processing stops, but those flags already processed remain in effect.

Some flags are toggled by specifying a single mnemonic, others use a one mnemonic; for the set condition and a another of the reset condition.

If replacements flags are omitted then the user is prompted for values.

flag mnemonics

Specifies one or more updated flag values for the **FLAGS** or **EFLAGS** registers.

The following mnemonics are defined. The value of *t* implies the flag value is toggled when the mnemonic is specified:

Flag	Bit	Value	Description
VM	17	t	Virtual 8086 Mode (EFLAGS only)
RF	16	t	Resume Flag - Disable Debug Exceptions (EFLAGS only)
NT	14	t	Nested Task
OV	11	1	Overflow
NV	11	1	¬ Overflow
DN	10	1	Direction Down
UP	10	0	Direction Up
EI	9	1	Enable Interrupts
DI	9	0	Disable Interrupts
NG	7	1	Negative Sign
PL	7	0	Plus Sign
ZR	6	1	Zero Result

Flag	Bit	Value	Description
NZ	6	0	Non-zero Result
AC	4	1	Auxiliary Carry
NA	4	0	¬ Auxiliary Carry
PE	2	1	Parity Even
PO	2	0	Parity Odd
CY	0	1	Carry
NC	0	0	¬ Carry

cr0 flag mnemonics

Specifies one or more updated flags values for the **CR0** register.

The following mnemonics are defined:

Bit	Value	Flag	Description
PG	31	1	Paging Enabled
ET	4	1	Extension Type Flag - x87 support
TS	3	1	Task Switch Flag
EM	2	1	Emulation exception
MP	1	1	Math Present
PM	0	1	Protect Mode Enabled

msw flag mnemonics

Specifies one or more updated flags values for the **MSW** register.

The following mnemonics are defined:

Flag	Bit	Value	Description
TS	3	1	Task Switch Flag
EM	2	1	Emulation exception
MP	1	1	Math Present
PM	0	1	Protect Mode Enabled

2-bit register

This option is used to specify that the **IOPL** field of the **FLAGS** or **EFLAGS** register should be updated with the specified replacement ***2-bit value***. The mnemonic **IOPL** is coded to specify this option.

If the replacement value is not specified then the user is prompted for a value.

16-bit register

This option is used to set the value of a register where *16-bit register* specifies either one of the standard INTEL register mnemonics or:

GDTL The GDT limit.

IDTL The IDT limit.

PC The program counter. This is synonymous with IP.

This option implies a request to update a register value. If the corresponding new 16-bit value is not specified then the prompted for a replacement value.

24-bit register
This option is used to set the base address of either the **GDT** or **IDT**. using **GDTB** and **IDTB** as mnemonics for these registers, respectively.

This option implies a request to update a register value. If the corresponding new 16-bit value is not specified then the prompted for a replacement value.

32-bit register
This option is used to set the value of a register where 16-bit register specifies one of the standard INTEL register mnemonics.

This option implies a request to update a register value. If the corresponding new 16-bit value is not specified then the prompted for a replacement value.

2-bit value
Specifies the 2-bit replacement value for the **IOPL**.

16-bit value
Specifies the 16-bit replacement value for a given 16-bit register.

24-bit value
Specifies the 24-bit replacement value for a given 24-bit register.

32-bit value
Specifies the 32-bit replacement value for a given 32-bit register.

Results and Notes:

The register information is stored in a special save area which the Kernel Debugger uses when entered and restores from this area when control returns to the system.

When no operands are specified the R command operates in display mode in exactly the same manner as the .R command.

When operands are specified, the R command operates in alter mode. If no replacement value is supplied on the command then the user is prompted with the current value followed by a colon prompt character. For example:

```
##R SS
0030
:
```

Flag register value prompts have their current flag setting interpreted using the mnemonics described above. For example:

```
##R EF
--(rf) --(vm) --(nt) nv(ov) up(dn) ei(di) pl(ng) nz(zr) na(ac) po(pe) nc(cy)
:
```

This example shows mnemonics for current settings followed by their negating mnemonic in brackets. For example:

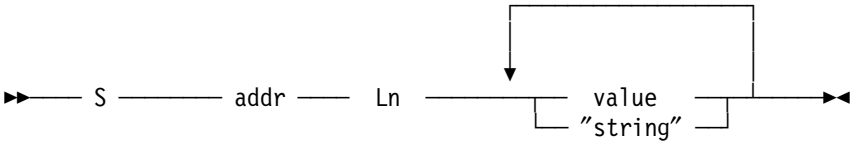
RF is not in effect, but since it is a toggle flag, the value **RF** specified at the prompt would set **RF**.

NV is in effect. To negate it, specify **OV** at the prompt.

3.3.36 S - Search Memory for Data

Search a memory range for occurrences of a list of bytes.

Syntax:



Parameters:

addr The address of the memory location to be searched.

Ln The number (n) of bytes to search.

value A numerical byte value to be searched into memory. One or more values may be specified separated commas or blanks. These may be mixed with *"**string**"* values.

string A character string enclosed in single or double quotes. Each character is treated as a list byte values to search memory, no terminating 0x00 value is stored. No folding of characters to upper or lower case occurs. One or more strings may be specified separated by commas or blanks. These may be mixed with numerical values.

Results and Notes:

The list of values and strings is used as a combined search argument. Only precise matches against the entire search argument are reported. The search is repeated for every byte location in the range specified. If no matches are found then nothing is displayed. Where matches are found the search command displays a list of storage addresses. For example:

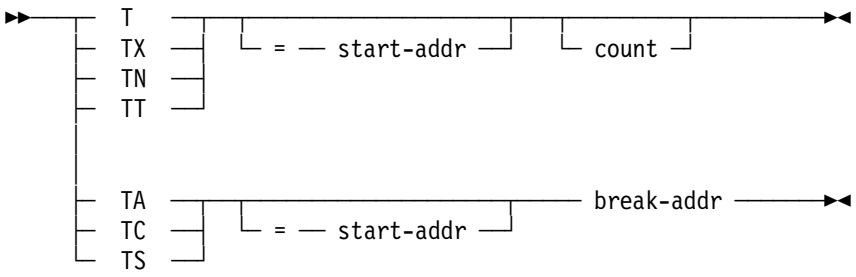
```
##s ptda_start l1000 "TD"
0030:0000fffe
ln 30:fffe
0030:0000fffe os2krnl:TASKAREA:ptda_signature
```

If memory is present, the storage is updated; otherwise an Invalid Address message is generated. If this should happen, valid storage may be paged into memory by means of the `.I` command.

3.3.37 T - Trace Instruction Execution

Trace instruction execution singly or for a specific number or instructions or to a specific address.

Syntax:



Parameters:

(Default) Trace one or more instructions, excluding known *bad* areas (see X subcommand below.)

A Trace all instructions to ***break-addr***.

This option requires ***break-addr*** to be specified.

C Counts all instructions executed until ***break-addr*** is reached.

Note: Counting is suspended when the system switches out of the current context in which the TC command was executed. It is resumed when that context switches back.

This option requires ***break-addr*** to be specified.

N Trace instructions but suppress the register display after each instruction is executed.

S The trace *special* option is similar to TC except that an intermediate instruction count is displayed before execution of each CALL instruction and after each return.

This option requires ***break-addr*** to be specified.

Notes: Counting is suspended when the system switches out of the current context in which the TS command was executed. It is resumed when that context switches back.

TS does not attempt to match CALL with RET instructions. Instead it inserts a temporary breakpoint at the instruction address following the CALL. In addition the TS command maintains a stack of return addresses and always checks the most recent two entries, as it single-instruction steps through the traced code, for a matching return address. This technique enables code that uses JMP instructions to return from a call to be better detected.

This is *not* a foolproof technique, especially where mutually recursive code is traced.

T This option causes the Kernel Debugger's trap vector handlers to be removed temporarily from the IDT and the system's re-instated until after then next instruction has executed. After execution of the next instruction the Kernel Debugger's V commands are re-instated.

This is a convenience option that saves manually unhooking a Kernel Debugger trap vector handlers from the IDT using a command sequence similar to:

```
VC n
T
VS n
```

X This option forces the Kernel Debugger to trace areas of system code that are known to be unsuitable for tracing. Normally, this occurs when **Trace** encounters one of the following routines:

```
_Debug_CtrlC32   through _EndCtrlC32
_DebugLoadSymMTE through EndDebugLoadSymMTE
_PGSwitchContext through pgSwitchRet
```

A temporary breakpoint is inserted at the routine's return address and the system is allowed to go to that address uninterruptedly. When TX is used the Kernel Debugger will attempt to trace instructions within these routines.

The consequence of forcing tracing in these routines may be at worst, the system is left in an unrecoverable state, and at *best* certain Kernel Debugger commands will give erroneous information.

start-addr The address from which execution is to continue. This must be a valid address for the current context. If ***start-addr*** is omitted then execution continues from the current **CS:EIP**, as shown by the R command.

> **Attention**
>
> Be very careful to ensure that the start address is valid for the privileged level and addressability of the code and data selectors in use. If the Kernel Debugger attempts to load a segment register that is invalid, the system may trap in the debugger code.

break-addr This is the address at which tracing will stop and the Kernel Debugger will be re-entered unless one of the following conditions is encountered:

A fatal exception occurs.

An Internal Processing Error (IPE) occurs.

A sticky breakpoint fires.

A non-maskable interrupt occurs.

An INT 3 instruction is executed.

The user enters Ctrl-C from the debugging console. The **break-addr** only remains in effect until the Kernel Debugger is next re-entered.

count This is the number of instructions to trace before re-entering the Kernel Debugger, unless one of the following conditions is encountered:

A fatal exception occurs.

An Internal Processing Error (IPE) occurs.

A sticky breakpoint fires.

A non-maskable interrupt occurs.

An INT 3 instruction is executed.

The user enters Ctrl-C from the debugging console.

If omitted then, ***count*** defaults to one instruction.

Results and Notes:

Except for TN, TC and TS the default command is executed when control returns to the debugging console. This defaults to the R command unless respecified through use of the ZS command.

TN suppresses the register display from the automatic R command, but still displays an unassembled next instruction for each traced instruction. If the ZS command has been used to specify a different default command then the TN command behaves exactly as T.

An example of the output from the TN command is as follows:

```
##TN 5
0170:fff4521f 803d9e53e0ffff cmp     byte ptr [InterruptLevel (ffe0539e)],ff
0170:fff45226 75b4           jnz     fff451dc
0170:fff45228 803d9643e0ff00 cmp     byte ptr [_cTKNoBlock (ffe04396)],00
0170:fff4522f 75be           jnz     fff451ef
0170:fff45231 0f01e1         smsw    cx
##
```

Note: The last traced instruction is the next to be executed.

TC displays the total number of instructions traced in the following message:

```
Total traced instructions: nnnn (decimal)
```

where ***nnnn*** is the number of traced instructions.

Following this message the default command is executed. See the Z command for details.

TS displays a variety of different messages, examples of which are:

```
------------------------------------------------------------------------
   Instruction Count: 101
d0df:0000f319 9a0000c810     call    10c8:0000
```

Accumulated number of instructions executed before the **CALL** instruction.

```
------------------------------------------------------------------------
Exit:
```

Accumulated number of instructions executed when the return address is encountered.

Note: This does not include the instruction at the return address.

```
-------------------------------------------------------------------------
 ...Special exit follows...
 Exit: 360
```

Accumulated number of instructions executed when the second most recent return address is encountered. In this case the most recent return address is discarded from the stack.

Note: This does not include the instruction at the return address.

```
-------------------------------------------------------------------------
Switching context...
   ...Back in context
```

Signifies context switching occurring and the suspension and resumption of instruction counting.

```
-------------------------------------------------------------------------
Total traced instructions: nnnn (decimal)
```

The total number of instructions traced when the ***break-addr*** is encountered.

Notes: REP and REPNE string instruction prefixes are handled differently to other instructions when single stepping. The Kernel Debugger generates a temporary breakpoint following the repeated string instructions (MOVS, CMPS, SCAS, LODS and STOS) and returns control to the system until the temporary breakpoint fires.

INT 3 instructions encountered when single-stepping are reported but in actual fact stepped over, thereby avoiding a double breakpoint at the same address.

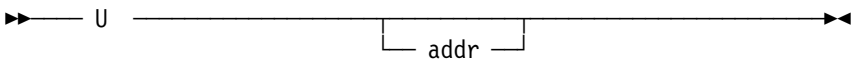
Attention

If any of the **Trace** commands is interrupted, the Kernel Debugger may leave a temporary breakpoint active. This will result in a **Trap 1** when the system is next given control. If this occurs then either of the TT or GT commands will clear this condition.

3.3.38 U - Unassemble

Unassemble storage at a given address.

Syntax:



Parameters:

addr The address of the storage location to be unassembled.

Results and Notes:

The U command unassembles storage from the address given. No attempt is made to distinguish between code and data storage. If no ***addr*** is given then the default address is determined in order of precedence as follows:

- The last unassembled address **+ 1**, or
- The **CS:EIP** of the last explicitly executed **.**R command, R command or
- The address of the next instruction to be executed.

Output from the U command is in two forms depending on whether the storage address was set in the context of the default (Kernel Debugger's or Dump Formatter's current) thread slot or another slot. In the former case output appears as in the following example:

```
##u
0170:fff4521f 803d9e53e0ffff cmp     byte ptr [InterruptLevel (ffe0539e)],ff
0170:fff45226 75b4           jnz     fff451dc
0170:fff45228 803d9643e0ff00 cmp     byte ptr [_cTKNoBlock (ffe04396)],00
0170:fff4522f 75be           jnz     fff451ef
0170:fff45231 0f01e1         smsw    cx
0170:fff45234 66f7c10200     test    cx,0002
0170:fff45239 0f8552050000   jnz     fff45791
0170:fff4523f fa             cli
```

In the latter case the context is shown by prefixing the thread slot to the address as in the following example:

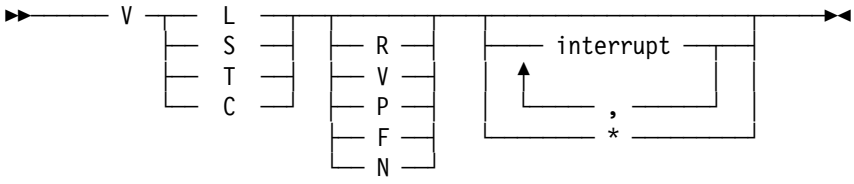
```
##.p*

 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
*0022# 0013 0003 0013 0001 blk 0300 7b6ea000 7b8c7128 7b8ab820 1eb8 18 epm
##.r 34

eax=00000000 ebx=000007f4 ecx=00000000 edx=0003ace7 esi=d02f4ef0 edi=000011ec
eip=0000272d esp=0000755e ebp=00007566 iopl=2 -- -- -- nv up ei ng nz na pe nc
cs=d02f ss=001f ds=bccf es=ace7 fs=150b gs=000  cr2=15b20000  cr3=001d9000
doscall1:CONFORM16:postDOSSEMWAIT:
0034|d02f:0000272d c9        leave
##u

doscall1:CONFORM16:postDOSSEMWAIT:
0034|d02f:0000272d c9        leave
0034|d02f:0000272e ca0800    retf    0008
0034|d02f:00002731 87db      xchg    bx,bx
0034|d02f:00002733 90        nop
doscall1:CONFORM16:DOSSEMSET:
0034|d02f:00002734 c8040000  enter   0004,00
0034|d02f:00002738 8b4608    mov     ax,word ptr [bp+08]
0034|d02f:0000273b 3d0200    cmp     ax,0002
0034|d02f:0000273e 7448      jz      2788
0034|d02f:00002740 250300    and     a,0003
0034|d02f:00002743 3d0100    cmp     ax,0001
0034|d02f:00002746 7415      jz      275d
0034|d02f:00002748 8b4608    mov     ax,word ptr [bp+08]
##
```

Note: The unassembled instruction mnemonics may be toggled between uppercase and lowercase by use of the Y command.

3.3.39 V - Exception/Trap/Fault Vector Commands

This group of commands manipulates IDT entries **0** through **e** to point to Kernel Debugger supplied interrupt handlers. By this means the Kernel Debugger may selectively be made to intercept each system exception before the system is allowed to process the exception. When a system exception is intercepted the Kernel Debugger gives control to the user. The original **IDT** entries are retained so that they may be re-instated, or given control following an exception which the Kernel Debugger has intercepted. See the GT and TT commands for information in returning control to the system exception handlers.

Syntax:



Parameters:

L The **List** subcommand list active Kernel Debugger trap and interrupt vectors.

Only a category specification (R, V, P, F or N) may be optionally specified with the **List** subcommand.

S The **Set** subcommand activates a Kernel Debugger exception vector according to criteria specified in the remaining parameters. Vectors set using this option cause the Kernel Debugger to receive control only when the corresponding exceptions are generated in ring 2 and 3 code.

T The **Trap** subcommand activates a Kernel Debugger exception vector according to criteria specified in the remaining parameters. Vectors set using this option cause the Kernel Debugger to receive control whenever the corresponding exceptions are generated regardless of the current privileged level.

C The **Clear** subcommand re-instates one or more system exception handlers according to the criteria specified in the remaining parameters.

R This option refines the exception criteria to real-mode exceptions only.

If no refining category is specified then the vector subcommand being executed applies to the R, V, P and F options simultaneously.

V This option refines the exception criteria to V86-mode exceptions only.

P This option refines the exception criteria to protect-mode exceptions only.

F This option refines the exception criteria to those exceptions that would be fatal to a process or the system. If a Local (system) exception handler is registered then the exception is not intercepted.

User exception handlers do not affect the operation of the Vector command. Local exception handlers protect the system from recoverable errors, in particular bad parameters passed in API calls. If a parameter causes the system to trap, the local exception handler is given control and the application is terminated. VSF will not intercept such traps. For further information on exception handling and how to intercept exceptions in general, see 1.6, “Trap and Exception Processing” on page 44.

N This option causes the Kernel Debugger exception handler to *beep* continuously instead of giving control to the user. The user may then break into the Kernel Debugger by entering Ctrl-C at the debugging console.

The **N** option works in conjunction with the four refining categories; that is, it does not by itself cause an interrupt to be trapped but instead specifies an action when that event occurs.

The **N** option must be explicitly specified for all four subcommands (**L S**, **T** and **C)** when required.

interrupt This allows one or more interrupt vectors, separated by commas, to be specified with the vector command as a refining criterion.

It is not valid with the **List** subcommand.

The abbreviation ***** may be specified as an alternative to the following interrupts, in each of the refining categories:

Real-mode: 0,1,2,3,4,5,6

V86-mode: 0,1,3,4,5,6,7,9,a,b,c,d,e

Protect-mode: 0,1,3,4,5,6,7,9,a,b,c,d,e

Fail option: 0,1,2,3,4,5,6,7,8,9,a,b,c,d,e

Noise option: 0,1,2,3,4,5,6,7,8,9,a,b,c,d,e

Results and Notes:

Only the **List** subcommand gives immediate output, which is of the form in the following example:

```
##VL
R 0 1 2 3 4 5 6
V
P d
F e d
```

As can be seen from this example, each category is shown with its one-letter abbreviation followed by a list of interrupt vectors currently being intercepted by the Kernel Debugger

Note: The **N** option must be specified explicitly to be listed.

All other subcommands only cause output to appear when an interrupt is intercepted. When this happens the following events occur:

1. The **N** option is checked; if enabled the Kernel Debugger emits a continuous beep until the user breaks in through the debugging console.

2. A trap message is issued if the default command is set to the R command.
3. The default command is executed.

The following figure shows the format of the trap messages issued by the Kernel Debugger exception handlers:

```
Trap 0 - Divide Error Exception
Trap 1 - Unexpected trace interrupt
Trap 2 - NMI Interrupt
Trap 4 - INTO Detected Overflow Exception
Trap 5 - BOUND Range Exceeded Exception
Trap 6 - Invalid Opcode Exception
Trap 7 - Processor Extension Not Available Exception
Trap 8 - Double Exception Detected nnnn
Trap 9 - Processor Extension Segment Overrun
Trap 10 (OAH) - Invalid TSS nnnn, mmmmmmmm
Trap 11 (OBH) - Segment Not Present nnnn, mmmmmmmm
Trap 12 (OCH) - Stack Segment Overrun or Not Present nnnn, mmmmmmmm
Trap 13 (ODH) - General Protection Fault nnnn, mmmmmmmm
Trap 14 (OEH) - Page Fault nnnn, mmmmmmm
```

In the messages above ***nnnn*** is substituted with the Intel exception code and ***mmmmmmmm*** is substituted with an interpretation of the Intel error code flags. For **Trap 10**, **Trap 11**, **Trap 12** and **Trap 13** the error code flags are interpreted as:

External External event

IDT Gate IDT gate selector error

GDT GDT selector error

LDT LDT Selector error

For **Trap 14** the error code flags are interpreted as a combination of:

Not Present Page not present

Read Access Read Access failure

Write Access Write Access Failure

User mode Fault occurred when executing in User mode

Supervisor Fault occurred when executing in Supervisor mode

If a trap occurs in the debugger component of the Kernel Debugger, the trap message will be appended with:

```
- In Debugger
```

If this happens then the only hope of recovering the system is to set the registers, using the R command, to a known consistent set of values.

See the *INTEL x86 Programmer's Reference* or the *INTEL Pentium User's Guide* for further information on exceptions and error codes.

Notes: **Trap 1** normally occurs as part of the operation of the Kernel Debugger. Therefore, only unexpected **Trap 1** exceptions are reported.

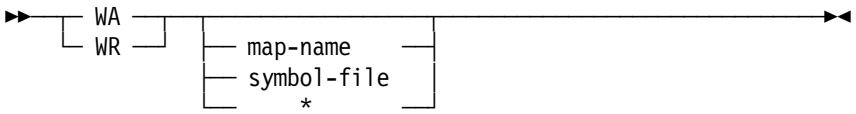
When a **Trap 1** is generated through the use of the Debug Registers, then the Kernel Debugger signals this with the message **Debug register hit**.

Trap 3 occurs through use of the INT 3 instruction. This is used both by the Kernel Debugger and user programs in implementing breakpoints. User programs may use the INT 3 instruction as a program controlled technique for breaking into the debugger. In these cases a trap message is not displayed.

3.3.40 W - Withmap Add/Remove

Add or remove a symbol map. Under the Kernel Debugger this merely activates or deactivates a symbol map. Under the Dump Formatter a symbol file may be re-loaded using the **Withmap** command.

Syntax:



Parameters:

A Activate 1 or all symbol maps.

Note: If the corresponding load module is not active then the map will remain deactivated. See the L command for more information on displaying map status.

R Remove 1 or all symbol maps.

L (not shown) This subcommand applied only to the Dump Formatter and has been superseded by the **.**LM command.

map-name The symbol map name to be activated or deactivated

symbol-file The symbol file name, with optional path and extension, to be loaded or removed.

Note: This operand applied only to the Dump Formatter.

***** Specifies all the maps or symbol files should be loaded or removed.

Results and Notes:

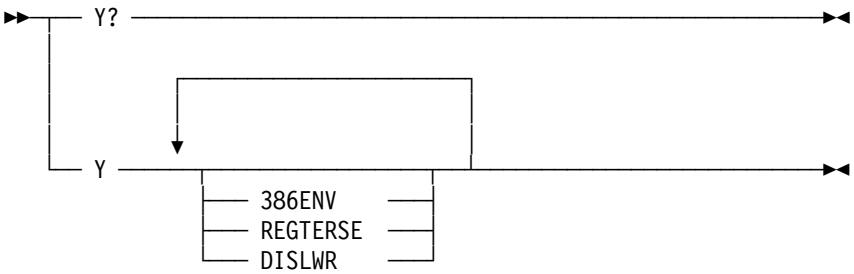
An error message is displayed only if the specified **map-name** is not loaded.

See the L command for related information.

3.3.41 Y - Set or Display Dump Formatter and Kernel Debugger Options

Set or display Dump Formatter and Kernel Debugger disassembly and register options.

Syntax:



Parameters:

(Default) Display current option settings.

? Display help for the **Y** command.

386ENV Force the Kernel Debugger and Dump Formatter to toggle the environment setting between 286 and 386 modes.

This affects the way in which commands interpret the register set. For example, in 286 mode, general registers are assumed, by default, to be 16-bit registers. Under rare circumstances it is necessary to force a particular mode to obtain a correct disassembly listing from the U command. Mostly this occurs in system code that is multi-modal and has juxtaposed sections of 32-bit and 16-bit code.

The initial setting is 386 mode under OS/2 V2.0 and above.

REGTERSE This has the same effect as the RT command.

The initial setting is for terse register display.

DISLWR This option toggles uppercase and lowercase display of assembler mnemonics from the U command.

The initial setting is for lowercase mnemonics.

Results and Notes:

No information is displayed when setting options.

When querying options, those in effect are displayed, for example:

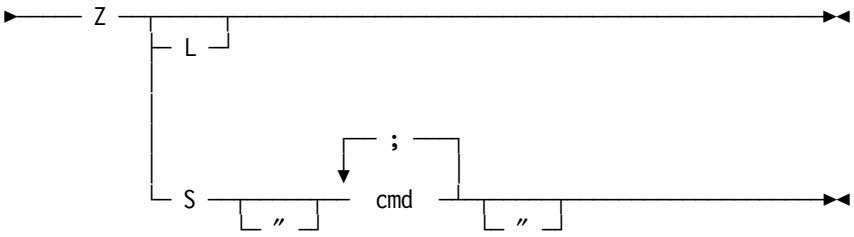
```
##y
386env dislwr regterse
```

This shows that the 386 environment is assumed, lowercase disassembly is in effect and terse register display is active. If any one of these settings is toggled, then the corresponding flag is not displayed.

3.3.42 Z - Set, Execute or Display the Default Command

Set, execute or display the default command.

Syntax:



Parameters:

(Default) Execute the default command string.

L Display the default command string

S Set the default command string.

cmd Specifies a Dump Formatter or Kernel Debugger commands to be used in the default command string. If the command string comprises more than one command, then each must be separated by commas and the entire string enclosed in quotes.

Results and Notes:

The default command string is executed automatically at breakpoints (where no other command string is associated with the breakpoint), after instruction tracing or when exception vectors are trapped. See the following commands for more information:

3.3.2, “B - Breakpoint Command Family” on page 84

3.3.25, “G - GO” on page 109

3.3.33, “P - PTrace Instruction Execution” on page 124

3.3.37, “T - Trace Instruction Execution” on page 133

3.3.39, “V - Exception/Trap/Fault Vector Commands” on page 139

When the Kernel Debugger and Dump Formatter are initialized the default command string is set to ″**R**″.

Note: When the user breaks into the Kernel Debugger with Ctrl-C the R command is executed regardless of the default command setting.

3.4 External Commands

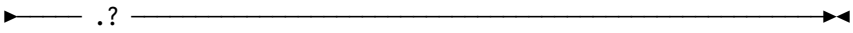
The following list comprises the set of external commands:

.?	Display external command help
.A	Display the SAS structure
.B	Set COM Parameters
.C	Display the Common ABIOS Data Area
.D	Display an OS/2 System Structure
.H	Display Dump File Header Information
.I (KDB)	Swap in Storage
.I (DF)	Display Dump State
.K	Display Ring 3 stack
.LM	Format Loader structures (MTE, OTE, STE)
.M	Formate Memory Structures
.MA	Format Memory Arena records (VMAR)
.MC	Format Memory Context Records (VMCO)
.MK	Format Memory Lock Information Records (VMLI)
.ML	Format Memory Alias Records (VMAL)
.MO	Format Memory Object Records (VMOB)
.MP	Format Memory Physical Page Frame Tables
.MV	Format Memory Virtual Frame Tables
.N	Display Dump Header Information
.P	Display Process Status
.PB	Display Blocked Thread Information
.PQ	Display Scheduler Thread Queuing Information
.PU	Display Thread User Space Information
.R	Display Ring 2/3 Registers
.REBOOT	Reboot the System Under Test
.S	Switch Default Thread Slot
.T	Format the System Trace Buffer

3.4.1 .? - Show External Command Help

Display help for internal Kernel Debugger and Dump Formatter commands.

Syntax:



Parameters:

None.

Results and Notes:

Displays a help summary for most of the Dump Formatter and Kernel Debugger external commands.

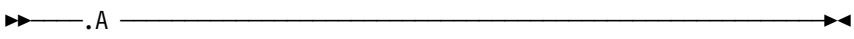
Note: Some of the information displayed is out-of-date.

Two pages of information are displayed with an intervening **--More--** prompt.

3.4.2 .A - Format the System Anchor Segment (SAS)

Format the System Anchor Segment (SAS).

Syntax:



Parameters:

None

Results and Notes:

The SAS is located from either GTD selector 70 or 78.

iod.A displays the following information:

```
--- SAS Base Section ---
                     SAS signature: SAS
           offset to tables section: 0016
      FLAT selector for kernel data: 0168
    offset to configuration section: 001E
    offset to device driver section: 0020
   offset to Virtual Memory section: 002C
          offset to Tasking section: 005C
              offset to RAS section: 006E
      offset to File System section: 0074
          offset to infoseg section: 0080
--- SAS Protected Modes Tables Section ---
                  selector for GDT: 0008
                  selector for LDT: 0000
                  selector for IDT: 0018
               selector for GDTPOOL: 0100
--- SAS Device Driver Section ---
    offset for the first bimodal dd: 0CB9
  offset for the first real mode dd: 0000
      sel for Drive Parameter Block: 04C8
       sel for ABIOS prot. mode CDA: 0000
        seg for ABIOS real mode CDA: 0000
                  selector for FSC: 00C8
--- SAS Task Section ---
          selector for current PTDA: 0030
  FLAT offset for process tree head: FFF10910
 FLAT address for TCB address array: FFF06BB6
      offset for current TCB number: FFDFFB5E
             offset for ThreadCount: FFDFFB62
--- SAS File System Section ---
                handle to MFT PTree: FE72CFBC
     selector for System File Table: 00C0
     sel. for Volume Parameter Bloc: 0788
   sel. for Current Directory Struc: 07B8
```

```
        selector for buffer segment: 00A8
--- SAS Information Segment Section ---
       selector for global info seg: 0428
   address of curtask local infoseg: 03C80000
      address of DOS task's infoseg: FFFFFFFF
         selector for Codepage Data: 07CB
--- SAS RAS Section ---
selector for System Trace Data Area: 04B0
 segment for System Trace Data Area: 04B0
        offset for trace event mask: 0B28
--- SAS Configuration Section ---
    offset for Device Config. Table: 0D50
--- SAS Virtual Memory Mgt. Section ---
       Flat offset of arena records: FFF13304
      Flat offset of object records: FFF1331C
     Flat offset of context records: FFF1330C
  Flat offset of kernel mte records: FFF0A891
     Flat offset of linked mte list: FFF07934
    Flat offset of page frame table: FFF11A70
    Flat offset of page range table: FFF111EC
    Flat offset of swap frame array: FFF03BAC
           Flat offset of Idle Head: FFF10090
           Flat offset of Free Head: FFF10080
          Flat offset of Heap Array: FFF11B78
     Flat offset of all mte records: FFF12E04
```

Each of the items displayed has the following significance:

--- SAS Base Section ---
: Marks the beginning of the **SAS** header section.

SAS signature
: **SAS** signature from **SAS_signature** (SAS+0x0). Always set to character value ″SAS ″.

offset to tables section
: Offset from SAS selector to the protected mode tables section.

  Taken from **SAS_tables_data** (SAS+0x4).

FLAT selector for kernel data
: Selector for 4G Read/Write addressability.

  Taken from **SAS_flat_sel** (SAS+0x6).

offset to configuration section
: Offset from SAS selector to the configuration tables section.

  Taken from **SAS_config_data** (SAS+0x8).

offset to device driver section
: Offset from SAS selector to the device driver section.

  Taken from **SAS_dd_data** (SAS+0xa).

offset to Virtual Memory section
: Offset from SAS selector to the Virtual Memory section.

  Taken from **SAS_vm_data** (SAS+0xc).

offset to Tasking section
: Offset from SAS selector to the Tasking section.

  Taken from **SAS_task_data** (SAS+0xe).

offset to RAS section
Offset from SAS selector to the RAS data section.

Taken from **SAS_RAS_data** (SAS+0x10).

offset to File System section
Offset from SAS selector to the File System section.

Taken from **SAS_file_data** (SAS+0x12).

offset to infoseg section
Offset from SAS selector to the Infoseg section.

Taken from **SAS_info_data** (SAS+0x1e).

--- SAS Protected Modes Tables Section ---
Marks the beginning of the protected mode tables section

selector for GDT
GDT selector that maps the GDT.

Taken from **SAS_tbl_GDT** (SAS_tables_section+0x0).

selector for LDT
No longer used.

Taken from **SAS_tbl_LDT** (SAS_tables_section+0x2).

selector for IDT
GDT selector that maps the IDT

Taken from **SAS_tbl_IDT** (SAS_tables_section+0x4).

selector for GDTPOOL
First GTD selector in selector pool. That is, first non-predefined GDT selector.

Taken from **SAS_tbl_GDTPOOL** (SAS_tables_section+0x6).

--- SAS Device Driver Section ---
Marks the beginning of the Device Driver Section

offset for the first bimodal dd
Offset from SAS selector to the first device driver header in the device driver chain.

See the **.**D command for formatting device driver headers.

Taken from **SAS_dd_bimodal_chain** (SAS_dd_section+0x0).

offset for the first real mode dd
No longer used.

Taken from **SAS_dd_real_chain** (SAS_dd_section+0x2).

sel for Drive Parameter Block
Selector that points to the head of the DPB chain.

See the **.**D command for formatting **DPBs**.

Taken from **SAS_dd_DPB_segment** (SAS_dd_section+0x4).

sel for ABIOS prot. mode CDA
Selector for ABIOS protect mode CDA.

See the **.**C command for displaying CDA information.

Taken from **SAS_dd_CDA_anchor_p** (SAS_dd_section+0x6).

seg for ABIOS real mode CDA

Segment for ABIOS real mode CDA. See the **.C** command for displaying CDA information.

Taken from **SAS_dd_CDA_anchor_r** (SAS_dd_section+0x8).

selector for FSC

Selector for the FSC segment.

Taken from **SAS_dd_FSC** (SAS_dd_section+0x2).

--- SAS Task Section ---

Marks the beginning of the tasking section.

selector for current PTDA

Selector for the current PTDA and ring 0 stack.

Taken from **SAS_task_PTDA** (SAS_task_section+0x0).

FLAT offset for process tree head

Linear address of **_pPTDAFirst**, which contains the linear address of the PTDA that heads the PTDA tree.

Taken from **SAS_task_ptdaptrs** (SAS_task_section+0x2).

FLAT address for TCB address array

Linear address of **_papTCBSlots**, which contains the linear address of the TCB array.

Taken from **SAS_task_threadptrs** (SAS_task_section+0x6).

offset for current TCB number

Linear address of **_TaskNumber**, which contains the current thread slot number.

Taken from **SAS_task_tasknumber** (SAS_task_section+0xa).

Offset for ThreadCount

Linear address of **_ThreadCount**, which contains the highest thread slot number in use - 1.

Taken from **SAS_task_threadcount** (SAS_task_section+0xe).

--- SAS File System Section --

Marks the beginning of the File System Section

handle to MFT PTree

Linear address of the head of the Ptree for the MFT.

See the .D command for formatting **MPTs**.

Taken from **SAS_file_MFT** (SAS_file_section+0x0).

selector for System File Table

Selector for the segment containing a table of selectors that point to tables of SFTs. Each SFT table contains an 8 byte header followed by contiguous SFT entries.

See the .D command for formatting **SFTs**.

Taken from **SAS_file_SFT** (SAS_file_section+0x4).

sel. for Volume Parameter Bloc

This is the selector for the work buffer used by volume mount processing.

Taken from **SAS_file_VPB** (SAS_file_section+0x6).

Note: The selector for the VPB segment is not given by this field. It may be located from the selector named by global variable **GDT_VPB**. See the **.**D command for formatting **VPBs**.

sel. for Current Directory Struc
Selector for the RMP segment containing CDS structures.

See the **.**D command for formatting **CDSs**.

Taken from **SAS_file_CDS** (SAS_file_section+0x8).

selector for buffer segment
Selector for the file system buffer segment.

Taken from **SAS_file_buffers** (SAS_file_section+0xa).

--- SAS Information Segment Section --
Marks the beginning of the Information Section.

selector for global info seg
Selector for the Global Information Segment (GISEG).

Taken from **SAS_info_global** (SAS_info_section+0x0).

address of curtask local infoseg
16:16 far pointer for the current Local Information Segment (LISEG).

Taken from **SAS_info_global** (SAS_info_section+0x2).

address of DOS task's infoseg
Real mode local information segment pointer (unused).

Taken from **SAS_info_localRM** (SAS_info_section+0x6).

selector for Codepage Data
Selector for the segment containing the Code Page Data Information Block (CDIB).

Taken from **SAS_info_CDIB** (SAS_info_section+0xa).

--- SAS RAS Section --
Marks the beginning of the RAS section

selector for System Trace Data Area
Selector for the STDA trace buffer.

Taken from **SAS_RAS_STDA_p**

See the **.**T command for formatting the system trace buffer. (SAS_RAS_section+0x0).

segment for System Trace Data Area
Selector for the STDA trace buffer.

Taken from **SAS_RAS_STDA_r** (SAS_RAS_section+0x2).

The same value is stored in both **SAS_RAS_STDA_p** and **SAS_RAS_STDA_r**.

offset for trace event mask
Offset from the **SAS** to the trace major event codes table (**ras_mec_table).**

Taken from **SAS_RAS_event_mask** (SAS_RAS_section+0x4).

--- SAS Configuration Section --
Marks the beginning of the Configuration section

offset for Device Config. Table
Offset from the **SAS** to the device configuration table.

Taken from **SAS_config_table** (SAS_config_section+0x0).

--- SAS Virtual Memory Mgt. Section --
Marks the beginning of the Virtual Memory Management section

Flat offset of arena records
The linear address of **_parvmOne**, the linear address of the first VM arena record (VMAR).

See the **.**MA command for related information.

Taken from **SAS_vm_arena** (SAS_vm_section+0x0).

Flat offset of object records
The linear address of **_pobvmOne**, the linear address of the first VM object record (VMOB).

See the **.**MO command for related information.

Taken from **SAS_vm_object** (SAS_vm_section+0x4).

Flat offset of context records
The linear address of **_pcovmOne**, the linear address of the first VM context record (VMCO).

See the **.**MC command for related information.

Taken from **SAS_vm_context** (SAS_vm_section+0x8).

Flat offset of kernel mte records
The linear address of **_DosModMTE**, the kernel (DOSCALLS.DLL) MTE.

See the **.**LM command for related information.

Taken from **SAS_vm_krnl_mte** (SAS_vm_section+0xc).

Flat offset of linked mte list
The linear address of **_global_h**, the linear address of the head of the MTE chain of link library modules.

See the **.**LM command for related information.

Taken from **SAS_vm_glbl_mte** (SAS_vm_section+0x10).

Flat offset of page frame table
The linear address of **_pft**, the linear address of the first (frame 0) page frame structure (PF).

See the **.**MP command for related information.

Taken from **SAS_vm_pft** (SAS_vm_section+0x14).

Flat offset of page range table
The linear address of **_pgPageablePAI**, the pageable PAI. The first double word of the PAI points to the page range table.

Taken from **SAS_vm_prt** (SAS_vm_section+0x18).

Flat offset of swap frame array
The linear address of **_smbmDF**, the linear address of swap frame allocation bit map followed by **_smFileSize**, the swap file size word length value in pages.

Taken from **SAS_vm_swap** (SAS_vm_section+0x1c).

Flat offset of Idle Head

The linear address of **_pgIdleList**, which points to the pseudo-PF at the head of the idle PF list.

See 3.4.17.2, “Idle Page Frame Structures” on page 227 for more information locating Idle Page Frame Structures.

See the **.**MP command for related information.

Taken from **SAS_vm_idle_head** (SAS_vm_section+0x20).

Flat offset of Free Head

The linear address of **_pgFreeList**, which points to the pseudo-PF at the head of the free PF list.

See the 3.4.17.1, “Free Page Frame Structures” on page 226 for more information locating free Page Frame Structures.

See the **.**MP command for related information.

Taken from **SAS_vm_free_head** (SAS_vm_section+0x24).

Flat offset of Heap Array

The linear address of **_apkh**, the array of VMKH kernel heap header structures. Note: the first entry is unused.

Taken from **SAS_vm_heap_info** (SAS_vm_section+0x28).

Flat offset of all mte records

The linear address of **_mte_h**, which is the linear address of the head of the MTE chain.

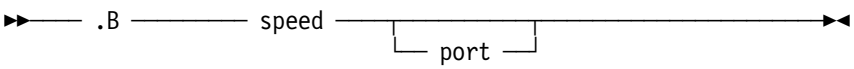
See the **.**LM command for related information.

Taken from **SAS_vm_all_mte** (SAS_vm_section+0x2c).

3.4.3 .B - Select the Communications Port and Speed

Select the communications port and speed.

Syntax:



Parameters:

Speed The **COMx** port speed. Any of the following values are valid:

150t

300t

600t

1200t

2400t

4800t

9600t

19200t

Note: Since baud rates are usually expressed in decimal and the default number base for the Kernel Debugger is hexadecimal, a **t** subscript must be supplied when using decimal values.

Port Specifies which COM port is to be used. If **1** or **2** are specified then COM1 or COM2 are implied. Any other numeric value is assumed to be an I/O port address.

Results and Notes:

When the Kernel Debugger initializes a default baud rate of **9600t** is set.

The COM port defaults to COM2 if there are two serial ports; otherwise the default is COM1, unless no COM ports are defined in the ROM BIOS data area, in which case the first port address in the ROM BIOS data area is assumed.

The parity, data and stop bit settings default to none, 8 and 1. These may be altered either:

From the Kernel Debugger by writing directly to the COM port control register using the **.**0 command.
From the system under test by using the MODE command.

If synchronization is lost with the debugging console, for example because the debugging communications port has been temporarily used by another application then it may be reset using the MODE command, from the command line of the system under test. For example, to re-specify the default parameters use:

```
MODE COM2 9600,n,8,1
```

3.4.4 .C - Display the Common ABIOS Data Area

Display ABOIS Command Data Area information.

Syntax:

Parameters: None

Results and Notes:

.C displays data for each logical device ID anchored from the Common ABIOS Data Area (CDA). If the ABIOS is not present or initialized then the following message is displayed:

```
ABIOS Not Present or Not Initialized
```

The presence of ABIOS is indicated by a non-zero byte value located at the symbol:

ABIOS_Present.

If the ABIOS is present and initialized, then data based on the Logical Device ID (LID) table is displayed. The LID Table is located from a selector located at:

ABIOS_CDS_ANCHOR_p - in protect mode

ABIOS_CDS_ANCHOR_r - in real mode

Tabular data of the following form is displayed:

```
LID(0000)  Type=Reserved     DB=001e:0114  FTT=0000:0000
LID(0001)  Type=Null         DB=0000:0000  FTT=0000:0000
LID(0002)  Type=Internal     DB=0438:06f0  FTT=0448:011c
LID(0003)  Type=Diskette     DB=0438:0728  FTT=0448:012c
LID(0004)  Type=Video        DB=0438:07a4  FTT=0448:017c
LID(0005)  Type=Keyboard     DB=0438:07e4  FTT=0448:01e4
LID(0006)  Type=Printer      DB=0438:080c  FTT=0448:0238
LID(0007)  Type=Asynch       DB=0438:082c  FTT=0448:0280
LID(0008)  Type=SysTimer     DB=0438:084c  FTT=0448:02e8
LID(0009)  Type=RTCTimer     DB=0438:0860  FTT=0448:0328
LID(000a)  Type=SysService   DB=0438:087c  FTT=0448:0380
LID(000b)  Type=NMInterrupt  DB=0438:08a0  FTT=0448:03cc
LID(000c)  Type=PointDevice  DB=0438:08d8  FTT=0448:0404
LID(000d)  Type=DMA          DB=0438:08f0  FTT=0448:044c
LID(000e)  Type=Security     DB=0438:0920  FTT=0448:04a4
LID(000f)  Type=POS          DB=0438:0938  FTT=0448:04f0
LID(0010)  Type=CMOSRam      DB=0438:0960  FTT=0448:0538
LID(0011)  Type=ErrorLog     DB=0438:0978  FTT=0448:0574
LID(0012)  Type==  DB=0438:0990  FTT=0448:05ac
LID(0013)  Type=Disk         DB=0438:09d8  FTT=0448:060c
LID(0014)  Type=anonymous)  DB=0438:0a50  FTT=0448:0684
LID(0015)  Type=Null         DB=0000:0000  FTT=0000:0000
LID(0016)  Type=Null         DB=0000:0000  FTT=0000:0000
LID(0017)  Type=Null         DB=0000:0000  FTT=0000:0000
```

```
LID(0018)  Type=Null        DB=0000:0000  FTT=0000:0000
LID(0019)  Type=Null        DB=0000:0000  FTT=0000:0000
LID(001a)  Type=Null        DB=0000:0000  FTT=0000:0000
LID(001b)  Type=Null        DB=0000:0000  FTT=0000:0000
LID(001c)  Type=Null        DB=0000:0000  FTT=0000:0000
LID(001d)  Type=Null        DB=0000:0000  FTT=0000:0000
```

Note: There is a formatting error that is illustrated in **LID 12** and **LID 14** lines. See description below of **Type=** parameter for an explanation of this!

The fields displayed have the following meaning:

LID Logical Device ID.

This is a sequential numbering of entries that appear in the table of LID entries. The entry, LID(0000), is however a dummy entry mapped by CDAType where the selector:offset of DB= are number of LID entries and offset to table of data pointers. Data pointer entries have one of the following forms:

Field Name	Offset	Length	Type	Description
DataPtr	+ 0	6		Data Pointer in CDA
DLimit	+ 0	2	W	Limit Field
DOffset	+ 2	2	W	Offset Field
DSegment	+ 4	2	W	Segment Field

Field Name	Offset	Length	Type	Description
PhysPtr	+ 0	6		Physical Data Pointer (INTEL Format)
	+ 0	2	W	Limit Field
PhysLSW	+ 2	2	W	Lo Order 16 bits
PhysMSW	+ 4	2	W	Hi Order 16 bits

Type= This an interpretation of the device type (**DevID**) field taken from the corresponding device block. The following Type values may appear:

Reserved	Used only for the LID(0000) dummy entry.
Null	signifies an unused entry (DB=0000:0000).
Internal	Devid=0000 used for internal ABIOS calls.
Diskette	Devid=0001 Diskette device.
Disk	Devid=0002 Disk device.
Video	Devid=0003 Video device.
Keyboard	Devid=0004 Keyboard.
Printer	Devid=0005 Printer.
Asynch	Devid=0006 Asynchronous device.
SysTimer	Devid=0007 System Timer.
RTCTimer	Devid=0008 RTC Timer.
SysService	Devid=0009 SysService.
NMInterrupt	Devid=000a NMI Interrupt.
PointDevice	Devid=000b Pointer Device.
LightPen	Devid=000c Light Pen.
JoyStick	Devid=000d JoyStick.
CMOSRam	Devid=000e CMOS RAM.

DMA	Devid=000f DMA controller.
POS	Devid=0010 Programmable Option Select.
ErrorLog	Devid=0011 Error Log.
S/A Dump	Devid=0012 Stand-Alone Dump.
IOPortAlloc.	Devid=0013 I/O Port Allocation.
Audiotone	Devid=0014 Audio device.
Int/8259	Devid=0015 Interrupt Controller.
Security	Devid=0016 Keyboard Security.

Other device types are in use but are not translated to a predictable name.

For example:

Devid=0017	SCSI Subsystem Interface
Devid=0018	SCSI Peripheral

Where this occurs the Devid may be found at offset +8 of the device block.

DB=sel:off sel:off address of the Device Block for the corresponding LID. The device block has the following standard header structure:

Field Name	Offset	Length	Type	Description
DeviceBlock	+ 0	8		Device Block Header
DevBlength	+ 0	2	W	Device Block Length
Revision	+ 2	1	B	Revision
	+ 3	1	B	Reserved
	+ 4	2	W	Logical ID
Devid	+ 6	2	W	Device ID

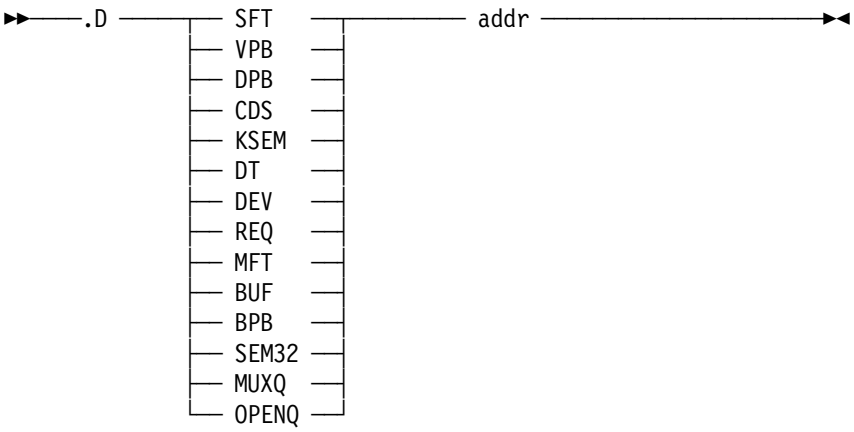
FTT=sel:off sel:off address to the Function Transfer Table for this LID. The FTT has the following standard header structure:

Field Name	Offset	Length	Type	Description
FTTable	+ 0	16		Function Transfer Table Header
FStart	+ 0	4	D	Start Routine Entry Point
FInt	+ 4	4	D	Interrupt Routine Entry Point
FTimeO	+ 8	4	D	Start Routine Entry Point
FuncCount	+ c	2	W	Count of Functions
	+ e	2	W	Reserved

3.4.5 .D - Display an OS/2 System Structure

Display an OS/2 System Structure.

Syntax:



Parameters:

structure

The structure type may take one of the following values:

SFT	Format a file system System File Table entry.
VPB	Format a file system Volume Parameter Block.
DPB	Format a file system Drive Parameter Block.
CDS	Format a file system Current Directory Structure.
KSEM	Format a Kernel Semaphore.
DT	Disk Trace in now obsolete.
DEV	Format a device driver header.
REQ	Format a device driver request packet.
MFT	Format a Master File Table entry.
BUF	Format a file system I/O buffer.
BPB	Format a BIOS Parameter Block.
SEM32	Format a 32-bit semaphore.
MUXQ	Format a **mutex** semaphore wait queue.
OPENQ	Format a 32-bit semaphore open queue.

addr

Specifies the address of the structure to be formatted. If omitted then the current **DS** selector value, offset **0** is assumed.

An address expression may be specified.

Results and Notes:

Attention

`.D` will format OS/2 structures without any validation. It is entirely incumbent on the user to ensure that the address used does in fact point to the named structure. Failure to observe this caution will result in meaningless information being displayed.

The following are examples of each of the 13 formatted structures. Refer to the System Reference for a description of each formatted structure.

SFT System File Table Entry
VPB Volume Parameter Block
DPB Drive Parameter Block
CDS Current Directory Structure
KSEM Kernel Semaphore
DEV Device Driver Header
REQ Device Driver (Strategy 1) Request Packet
MFT Master File Table Entry
BUF File System Buffer
BPB BIOS Parameter Block
SEM32 32-Bit Semaphore
MUXQ Semaphore MUX Queue
OPENQ Semaphore Open Queue

3.4.5.1 System File Table Entry (SFT)

```
.d sft d0:8
      sf_ref_count: 0001                      sfi_mode: 00a0
        sf_usercnt: 0000                      sfi_hVPB: 0012
          reserved: 00                       sfi_ctime: 0000
       sf_flags(2): 0100:0000                sfi_cdate: 0000
         sf_devptr: #0000:0000               sfi_atime: 0000
            sf_FSC: #00c8:0008               sfi_adate: 0000
          sf_chain: #0000:0000               sfi_mtime: 0000
            sf_MFT: fe7fb788                 sfi_mdate: 0000
sfdFAT_firFILEclus: 5ad6                      sfi_size: 000bb135
    sfdFAT_cluspos: 09c8                   sfi_position: 00085d90
    sfdFAT_lstclus: 0000                        sfi_UID: 0000
     sfdFAT_dirsec: 00000000                    sfi_PID: 0000
     sfdFAT_dirpos: 00                          sfi_PDB: 0000
       sfdFAT_name:                       sfi_selfsfn: 0000
   sfdFAT_EAHandle: 0000                       sfi_tstamp: 00
         sf_plock: 0000                      sfi_DOSattr: 20
      sf_NmPipeSfn: 0000
       sf_codepage: 0000
##
```

Figure 28. System File Table Entry

Notes: The **sfdFAT_name** is only meaningful for SFTs that represent open FAT file system files.

For a description of the SFT fields see the System File Table Entery (SFT) in the System Reference.

3.4.5.2 Volume Parameter Block (VPB)

```
##ln gdt_vpb
0138:00000098 os2krnl:DOSGDTDATA:GDT_VPB
##.d vpb 98:12
        vpb_flink: 0000                  vpdFAT_cluster_mask: 41
        vpb_blink: 008d                 vpdFAT_cluster_shift: 00
    vpb_ref_count: 007b                     vpdFAT_first_FAT: 00b2
 vpb_search_count: 0000                     vpdFAT_FAT_count: a8
 vpb_first_access: 00                    vpdFAT_root_entries: 0004
    vpb_signature: 444a                  vpdFAT_first_sector: 885c0400
     vpb_flags(2): 02:00                  vpdFAT_max_cluster: 410e
          vpb_FSC: #00c8:0008                vpdFAT_FAT_size: b200
           vpi_ID: 26715015               vpdFAT_dir_sector: aa04a800
         vpi_pDPB: #04b8:00c4                    vpdFAT_media: 0d
     vpi_cbSector: 0200                    vpdFAT_next_free: 00b2
       vpi_totsec: 0007cfe0                 vpdFAT_free_cnt: 04a8
       vpi_trksec: 0020                  vpdFAT_FATentrysize: b2
        vpi_nhead: 0040                      vpdFAT_IDsector: 00000000
         vpi_pDCS: #0000:0000                  vpdFAT_access: 0000
         vpi_pVCS: #0000:0000                 vpdFAT_accwait: 0000
        vpi_drive: 07                          vpdFAT_pEASFT: #0000:0000
         vpi_unit: 07
         vpi_text: UNLABELED
         vpi_flags: 0003
####.m 98:0

*har     par      cpg        va    flg next prev link hash hob   hal
 0003 %feaef04c 00000400 %fe6ef000 001 0002 0023 0000 0000 0003 0000      =0000
 hob   har hobnxt flgs own  hmte  sown,cnt lt st xf
 0003  0003 fec5  0000 ffec 0000  0000 00  01 00 00 vmkrhrw
   pvmli    cs    eip     phlock   cpg      va    flg  hptda hob sig  csig
 %fe82e4c4 002d 0a6800a5 %ac22403c 0001 %fe83c000 0005 024b 0003 ea9f ea9f
##dd %(98:0)-10 l8
%fe6fd4dc  00000001 ff5905b8 5e02bd64 000009bd
%fe6fd4ec  05d6007b 000009ae 0000099c dac40000
##dd %(98:0)-10-4+9bc l8
%fe6fde94  6d6b6d62 00180000 ffa20098 00e800e8
%fe6fdea4  ffc2001c 00000008 00000000 00000000
##.mo ffa2
ffa2 vpb
##
```

Figure 29. Volume Parameter Block

Notes:

The selector for the **VPB** segment may be found by listing the symbol **GDT_VPB** and using its offset.

The handle of a **VPB** (**hVPB**) is the offset within the **VPB** segment.

The VPB segment has a unique owner ID, which may be determined using the **.M** command. In the case of the VPB segment it is allocated from the kernel resident heap, so the true owner id is found in the heap header (or its extension - the trailer).

For a description of the VPB fields see the Volume Parameter Block (VPB) in the System Reference.

3.4.5.3 Drive Parameter Block (DPB)

```
.d dpb 4b8:c4
        dpb_drive: 07
         dpb_unit: 07
  dpb_driver_addr: #0798:0000
     dpb_next_dpb: #04b8:00e0
     dpb_cbSector: 0200
    dpb_first_FAT: 0001
  dpb_toggle_time: 00000000
         dpb_hVPB: 0012
        dpb_media: f8
        dpb_flags: 20
   dpb_drive_lock: 0000
    dpb_strategy2: #07a0:139c


##.m 04b8:0c4

*har     par      cpg        va    flg next prev link hash hob   hal
 0003 %feaef04c 00000400 %fe6ef000 001 0002 0023 0000 0000 0003 0000      =0000
 hob   har hobnxt flgs own  hmte  sown,cnt lt st xf
 0003  0003 fec5  0000 ffec 0000  0000 00  01 00 00 vmkrhrw
   pvmli    cs    eip     phlock   cpg      va    flg  hptda hob sig  csig
 %fe82e4c4 002d 0a6800a5 %ac22403c 0001 %fe83c000 0005 024b 0003 ea9f ea9f
##dd %(4b8:0) - 10 l8
%fe6f1638  00000000 ff360498 2000fe6f 000001c5
%fe6f1648  00000000 001c0798 020004b8 00000001
##dd %(4b8:0) - 10-4+1c4 l8
%fe6f17f8  00000000 00000000 ff9604b8 0000000a
%fe6f1808  ff46000c 000014f8 00d020c8 ff46000c
##.mo ff96
ff96 dpb
```

Figure 30. Drive Parameter Block

Notes:

The **DPB** may be located from the **VPB**.

The DPB segment has a unique owner ID, which may be determined using the **.M** command. In the case of the VPB segment it is allocated from the kernel resident heap, so the true owner ID is found in the heap header (or its extension - the trailer).

For a description of the DPB fields see the Driver Parameter Block (DPB) in the System Reference.

3.4.5.4 Current Directory Structure (CDS)

```
>> locate the SAS file system section
##dw 70:12 l1
0070:00000012  0074
##dw 70:74
0070:00000074  0fa4 fe70 00c0 07f8 0828 00a8 0428 0000
0070:00000084  03c8 ffff ffff 0843 0000 0000 0000 0000
0070:00000094  0000 0000 0000 0000 0000 0000 0000 0000
0070:000000a4  0000 0000 0000 0000 0000 0000 0000 0000
0070:000000b4  0000 0000 0000 0000 0000 0000 0000 0000
0070:000000c4  0000 0000 0000 0000 0000 0000 0000 0000
0070:000000d4  0000 0000 0000 0000 0000 0000 0000 0000
0070:000000e4  0000 0000 0000 0000 0000 0000 0000 0000

>> +8 into the file system section is the CDS RMP selector.
>> Can verify this by checking out the memory object owner.

##.m 828:0

*har     par      cpg        va    flg next prev link hash hob   hal
 0003 %feaef04c 00000400 %fe6ef000 001 0002 0023 0000 0000 0003 0000      =0000
 hob   har hobnxt flgs own  hmte  sown,cnt lt st xf
 0003  0003 fec5  0000 ffec 0000  0000 00  01 00 00 vmkrhrw
   pvmli    cs    eip     phlock   cpg      va    flg  hptda hob sig  csig
 %fe82e4c4 002d 0a6800a5 %ac22403c 0001 %fe83c000 0005 024b 0003 ea9f ea9f

>> Owned by the Kernel Resident Heap. Look at the header

##dd %(828:0)-10 l8
%fe7015b4  000007d0 ff5c07b0 0000bd64 0000060d
%fe7015c4  049d0600 01ae0014 00000001 00000400

>> This is an attributed block so look at the trailer

##dd %(828:0)-10-4+60c l8
%fe701bbc  00000000 00000000 ff610828 0000fe70
%fe701bcc  ff9e0054 4d45534b 00000201 00000000

##.mo ff61
ff61 cdsrmp

>> 828 does indeed point to the CDS RMP


>> Now dump the CDS handle table for the process of interest
##.p#
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
 0048# 0029 0004 0029 0001 blk 0200 ab805000 ab99b820 ab97fc20 1ed4 11 cmd

##dw %ab99b820 +cds_handle-ptda_start l1a
%ab99bac8  0000 0000 0000 0000 0000 0000 0000 0090
%ab99bad8  0000 0000 0000 0000 0000 0000 0000 0000
%ab99bae8  0000 0000 0000 0000 0000 0000 0000 0000
%ab99baf8  0000 0000
```

Figure 31. (Part 1 of 2). Current Directory Structure

```
>> Except for driver 07 (H:) the current directory handle is null.
>> This implies that the current directory for drive H: is not the
>> root. To see which it is, we need to locate the the CDS entry with
>> handle 0x0090.


>> The RMP has a 0x14 byte header. Each entry is prefixed with a word
>> length followed by the handle for that entry.

>> Starting with the first entry scan through until handle 0x0090 is
>> located.

##dw 828:14 l4
0828:00000014  8028 0000 00b2 0000

##dw 828:14 l2
0828:00000014  8028 0000

##dw 828:14+28 l2
0828:0000003c  0028 001c

##dw 828:14+28+28 l2
0828:00000064  0028 001d

##dw 828:14+28+28+28 l2
0828:0000008c  0026 001e

##dw 828:14+28+28+28+26 l2
0828:000000b2  8023 0014

##dw 828:14+28+28+28+26+23 l2
0828:000000d5  0028 0022

##dw 828:14+28+28+28+26+23+28 l2
0828:000000fd  002e 008f

##dw 828:14+28+28+28+26+23+28+2e l2
0828:0000012b  0025 0090

>> The CDS starts after the length prefix.

##.d cds 828:12b+2
   cd_handle: 0090                          cddFAT_id: 0000
   cd_refcnt: 0002
    cd_flags: 40
   cd_devptr: 04b8:00c4
 cd_OwnerFSC: 0008

    cdi_hVPB: 0012
     cdi_end: 0002
   cdi_flags: 80
    cdi_text: H:\spool
##
```

Figure 32. (Part 2 of 2). Current Directory Structure

Notes:

The selector for the **CDS** segment may be located from the SAS, as illustrated above, or form the storage at label **CDSAddr**.

The CDS RMP has a unique owner ID, which may be determined using the **.**M command. In the case of the CDS RMP, it is allocated from the kernel resident heap, so the true owner id is found in the heap header (or its extension - the trailer).

For a description of the CDS fields see the Current Directory Structure (CDS) in the *System Reference*.

3.4.5.5 Kernel Semaphore (KSEM)

```
>> Intra-Process serialisation mutex KSEM imbedded in the PTDA
.p#
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
*000c# 0002 0000 0002 0004 blk 0804 ab78d000 ab997020 ab978420 1c9c 00 cntrl
##.d ksem %ab997020 +ptda_ptdasem-ptda_start
Signature     : KSEM                        Nest: 0000
Type          : MUTEX
Flags         : 00
Owner         : 0000              PendingWriters: 0000
##


>> MFT Shared KSEM imbedded at the beginning of the MFT

##..d sft d0:8
      sf_ref_count: 0001                       sfi_mode: 00a0
        sf_usercnt: 0000                       sfi_hVPB: 0012
          reserved: 00                        sfi_ctime: 0000
       sf_flags(2): 0100:0000                 sfi_cdate: 0000
         sf_devptr: #0000:0000                sfi_atime: 0000
            sf_FSC: #00c8:0008                sfi_adate: 0000
          sf_chain: #0000:0000                sfi_mtime: 0000
            sf_MFT: fe7fb788                  sfi_mdate: 0000
sfdFAT_firFILEclus: 5ad6                      sfi_size: 000bb135
    sfdFAT_cluspos: 09c8                   sfi_position: 00085d90


##.d ksem %fe7fb788
Signature     : KSEM                        Nest: 0000
Type          : SHARE                    Readers: 0000
Flags         : 01                PendingReaders: 0000
Owner         : 0000              PendingWriters: 0000
##

>> Slot 49 is blocked. So we proceed by finding out what the BlockId
>> represents by finding its owner.

##.pb 49
 Slot  Sta BlockID  Name     Type        Addr    Symbol
 0049  blk fe83bdf4 warp_d

##.m  %0fe83bdf4

*har     par      cpg        va    flg next prev link hash hob   hal
 072c %feaf8dd2 00000400 %00540000 149 072d 072b 0003 0000 0003 0025 hptda=0878
 hal=0025 pal=%ffe5d140  har=072c  hptda=0878  pgoff=00000  f=021
 har     par      cpg        va    flg next prev link hash hob   hal
 0003 %feaef04c 00000400 %fe6ef000 001 0002 0023 0000 0000 0003 0000      =0000
 hob   har hobnxt flgs own  hmte  sown,cnt lt st xf
 0003  072c fec5  1000 ffec 0000  0000 00  01 00 00 vmkrhrw
   pvmli    cs    eip     phlock   cpg      va    flg  hptda hob sig  csig
 %fe82e4c4 002d 0a6800a5 %ac22403c 0001 %fe83c000 0005 024b 0003 ea9f ea9f
```

Figure 33. (Part 1 of 2). Kernel Semaphore

```
>> BlockID is in the kernel resident heap - assume that it is at the
>> beginning of a data portion of a heap block.
>> Dump the header.

##dd  %0fe83bdf4-10 l8
%fe83bde4  00000000 bd100000 fe83fe83 ff7e0018
%fe83bdf4  4d45534b 000a0004 46380001 bd28a801

>> Object Owner Id is ff7e
##.mo ff7e
ff7e ksem
##.d ksem %0fe83bdf4
Signature     : KSEM
Type          : EVENT
Flags         : 04
Owner         : 000a              PendingWriters: 0001
##
```

Figure 34. (Part 2 of 2). Kernel Semaphore

Notes:

KSEMs are usually found imbedded in system control blocks for serialization and sharing purposes.

Dynamically allocated KSEMs are allocated out of one of the kernel heaps.

Virtual Device Driver semaphore helper services result in KSEMs.

Under the ALLSTRICT kernel only, the KSEM has a signature field. This is manufactured by the **.D** command for non-ALLSTRICT kernels. Under the ALLSTRICT kernel the presence of a KSEM may be verified by dumping the KSEM in bytes. Offset **+0x0** is where the signature is located.

The owner field refers to the slot number of the semaphore owner.

For a description of the KSEM structure see the Kernel Semaphore Structure in the *System Reference*.

3.4.5.6 Physical Device Driver Header (DEV)

```
>> Driver header address taken from the VBP with handle 12:

.d vpb 98:12
        vpb_flink: 0000                  vpdFAT_cluster_mask: 41
        vpb_blink: 008d                vpdFAT_cluster_shift: 00
    vpb_ref_count: 007a                   vpdFAT_first_FAT: 00b2
 vpb_search_count: 0000                   vpdFAT_FAT_count: a8
 vpb_first_access: 00                  vpdFAT_root_entries: 0004
    vpb_signature: 444a                vpdFAT_first_sector: 885c0400
     vpb_flags(2): 02:00                vpdFAT_max_cluster: 410e
          vpb_FSC: #00c8:0008               vpdFAT_FAT_size: b200
           vpi_ID: 26715015               vpdFAT_dir_sector: aa04a800
         vpi_pDPB: #04b8:00c4                    vpdFAT_media: 0d
     vpi_cbSector: 0200                   vpdFAT_next_free: 00b2
       vpi_totsec: 0007cfe0                  vpdFAT_free_cnt: 04a8
       vpi_trksec: 0020                 vpdFAT_FATentrysize: b2
        vpi_nhead: 0040                     vpdFAT_IDsector: 00000000
         vpi_pDCS: #0000:0000                 vpdFAT_access: 0000
         vpi_pVCS: #0000:0000                vpdFAT_accwait: 0000
        vpi_drive: 07                         vpdFAT_pEASFT: #0000:0000
         vpi_unit: 07
         vpi_text: UNLABELED
        vpi_flags: 0003
##.d dpb 4b8:c4
        dpb_drive: 07
         dpb_unit: 07
  dpb_driver_addr: #0798:0000
     dpb_next_dpb: #04b8:00e0
     dpb_cbSector: 0200
    dpb_first_FAT: 0001
  dpb_toggle_time: 00000000
         dpb_hVPB: 0012
        dpb_media: f8
        dpb_flags: 20
   dpb_drive_lock: 0000
    dpb_strategy2: #07a0:139c

##.d dev 798:0
          DevNext: 0778:0000
          DevAttr: 2880
         DevStrat: 0dbc
           DevInt: 0000
         NumUnits: 0c
        DevProtCS: 07a0
        DevProtDS: 0798
        DevRealCS: 0000
        DevRealDS: 0000
```

Figure 35. (Part 1 of 2). Physical Device Driver Header

```
>> Formatting the device driver directly from the 1st data segment.

##.lmo ' ibmkbd
hmte=009b pmte=%fe6f1fb8 mflags=8008e1c9 h:\ibmkbd.sys
seg  sect psiz vsiz hob  sel  flags
0001 0001 0194 01fa 0000 0540 8d49 data iter prel rel
0002 0002 11c2 11c4 0000 0548 8d60 code shr prel rel
##.d dev 540:0
          DevNext: 0510:0000
          DevAttr: 8980
         DevStrat: 0680
           DevInt: 0586
          DevName: IBMKBD$
        DevProtCS: 0548
        DevProtDS: 0540
        DevRealCS: 0000
        DevRealDS: 0000
##
```

Figure 36. (Part 2 of 2). Physical Device Driver Header

Notes:

The Device Header appears in one of two formats, depending on whether the device supports multiple units or not.

The interrupt routine address is not used by OS/2.

For a description of the DEV fields see the Physical Device Driver Header (DEV) in the *System Reference*.

3.4.5.7 Device Driver (Strategy 1) Request Packet (REQ)

```
>> The two request packet pools for general device driver use:

##.moc 93

*har     par      cpg        va    flg next prev link hash hob   hal
 0091 %feaefc80 00000010 %ab546000 129 0090 0092 0000 0000 0093 0000   sel=04a8
 hob   har hobnxt flgs own  hmte  sown,cnt lt st xf
 0093  0091 0000  0124 ff40 0000  0000 00  00 00 00 reqpkt1
##.moc 94

*har     par      cpg        va    flg next prev link hash hob   hal
 0092 %feaefc96 00000010 %ab536000 129 0091 0093 0000 0000 0094 0000   sel=04b0
 hob   har hobnxt flgs own  hmte  sown,cnt lt st xf
 0094  0092 0000  0124 ff33 0000  0000 00  00 00 00 reqpkt2



>> Formatting the request packet assigned to slot 43:

##.p#
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
*0043# 000d 0004 000d 0005 blk 081f ab7fb000 ab99ac20 ab97f220 1d24 17 faxworks

##dd %ab97f220 +1ac l1
%ab97f3cc  04a80832

##.d req 4a8:832
           PktLen: 2c
          PktUnit: 00
           PktCmd: 04
        PktStatus: 0000
       PktDOSLink: 00000000
       PktDevLink: 00000000
          IOmedia: 00
          IOpData: 008f7883
          IOcount: 0000
          IOstart: 00000000
```

Figure 37. (Part 1 of 2). Device Driver Request Packets

```
>> Formatting the request packet assigned to slot 2:

##.p2
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
 0002  0001 0000 0000 0002 blk 0200 ab779000 ffe3ca00 ab977020 1f3c 00 *tsd
##dd %ab977020 +1ac l1
%ab9771cc  04a80012
##.d req 4a8:12
           PktLen: 32
          PktUnit: 00
           PktCmd: 00
        PktStatus: 0000
       PktDOSLink: 00000000
       PktDevLink: 00000000
        InitcUnit: 00
       InitDevHlp: 0000:0000
        InitParms: 0000:0000
          InitDrv: 00
     InitSysiData: 0000
##
```

Figure 38. (Part 2 of 2). Device Driver Request Packets

Notes:

Request Packets are allocated from one of three pools:

- Strategy 1 request pool
- Strategy 2 request pool
- Swapper request pool

Each thread is preassigned a strategy 1 request packet. If this is in use when a device driver tries to allocate another, then a packet is allocated from the strategy 2 pool for strategy 1 use.

Asynchronous read and write requests are implemented in DOSCALL1.DLL by creating multiple threads on which to run the parallel I/O requests.

`.D REQ` does not format Strategy 2 format Request Packets.

For a description of the Request Packet fields see the Device Driver Request Packer in the *System Reference*.

3.4.5.8 Master File Table Entry (MFT)

```
>> Display the SFT for SFN 20

##.d sft d0:(83*20+8)
      sf_ref_count: 0001                      sfi_mode: 00c2
        sf_usercnt: 0000                      sfi_hVPB: 0000
          reserved: 00                       sfi_ctime: 0000
       sf_flags(2): 00c0:0000                sfi_cdate: 0000
         sf_devptr: #0af0:0000               sfi_atime: 0000
            sf_FSC: #00c8:ff40               sfi_adate: 0000
          sf_chain: #00d0:170f               sfi_mtime: b19d
            sf_MFT: fe82ff7c                 sfi_mdate: 1f5f
sfdFAT_firFILEclus: 0000                      sfi_size: 00000000
    sfdFAT_cluspos: 0000                   sfi_position: 00000000
    sfdFAT_l


>> From the SFT display the MFT

.d mft %fe82ff7c
     mft_ksem:
Signature     : KSEM                          Nest: 0000
Type          : SHARE                      Readers: 0000
Flags         : 01                  PendingReaders: 0000
Owner         : 0000                PendingWriters: 0000
     mft_lptr: 0000                       mft_sptr: 00d0:08bb
    mft_pCMap: 00000000                   mft_serl: 013e       mft_signature: 466d
 mft_CMapKSem:
     mft_hvpb: 0000                  mft_opflags: 0000           mft_flags: 0000
     mft_name: \DEV\MOUSE$


>> Display the SFT for SFN 40

##.d sft d0:(83*40+8)
      sf_ref_count: 0001                      sfi_mode: 1302
        sf_usercnt: 0000                      sfi_hVPB: 008d
          reserved: 00                       sfi_ctime: 0000
       sf_flags(2): 0000:0000                sfi_cdate: 0000
         sf_devptr: #0000:0000               sfi_atime: 0000
            sf_FSC: #00c8:0008               sfi_adate: 0000
          sf_chain: #00d0:214b               sfi_mtime: 0000
            sf_MFT: fe6f190c                 sfi_mdate: 0000
sfdFAT_firFILEclus: 470c                      sfi_size: 00000000
    sfdFAT_cluspos: 09c8                   sfi_position: 00000000
    sfdFAT_l
```

Figure 39. (Part 1 of 2). Master File Table Entries

```
>> From the SFT display the MFT

.d mft %fe6f190c
     mft_ksem:
Signature     : KSEM                        Nest: 0000
Type          : SHARE                    Readers: 0000
Flags         : 01                PendingReaders: 0000
Owner         : 0000              PendingWriters: 0000
     mft_lptr: 0000                    mft_sptr: 00d0:2045
    mft_pCMap: 00000000                mft_serl: 00a3       mft_signature: 466d
 mft_CMapKSem:
     mft_hvpb: 008d                 mft_opflags: 0000           mft_flags: 0001
     mft_name: D:\SWAPPER.DAT
##
```

Figure 40. (Part 2 of 2). Master File Table Entries

Notes:

The MFT is entry may be located from each SFT that represents an open instance of a file.

The MFT points to the most recent SFT open instance of the file.

For a description of the MFT field, see the Master File Table Entry (SFT) in the *System Reference*.

3.4.5.9 File System Buffer (BUF)

```
>> Locate the file system buffer segment

##ln gdt_buffers
0138:000000a8 os2krnl:DOSGDTDATA:GDT_Buffers

##dw a8:0
00a8:00000000  ade4 94c4 0000 ff93 005a 0218 bc8b 0000
00a8:00000010  0664 0200 bc8c 000f 9000 00e5 0234 ade4
00a8:00000020  0000 0279 e05e 0000 0001 0400 0000 0000
00a8:00000030  0000 0000 46e5 4e49 3030 3630 4d54 2050
00a8:00000040  0000 0000 0000 0000 0000 b5cd 1f5e 3251
00a8:00000050  0400 0000 46e5 4e49 3030 3730 4d54 2050
00a8:00000060  0000 0000 0000 0000 0000 b5c8 1f5e 0000
00a8:00000070  0000 0000 4de5 3046 3030 2038 4d54 2050

##.d buf a8:ade4
   buf_next: 001c           buf_prev: ffff           buf_freeLink: 0000
  buf_flags: 02             buf_hVPB: 0279           buf_sector: 00000001
    buf_tid: 00             buf_wrtcnt: 02           buf_wrtcntinc: 0096
   buf_fill: 0000
##.d buf a8:234
   buf_next: 044c           buf_prev: 001c           buf_freeLink: 0000
  buf_flags: 04             buf_hVPB: 0279           buf_sector: 0000e05d
    buf_tid: 00             buf_wrtcnt: 01           buf_wrtcntinc: 0000
   buf_fill: 0000
##.d buf a8:44c
   buf_next: 0664           buf_prev: 0234           buf_freeLink: 0000
  buf_flags: 02             buf_hVPB: 0279           buf_sector: 0000001c
    buf_tid: 00             buf_wrtcnt: 02           buf_wrtcntinc: 0096
   buf_fill: 0000
##.d buf a8:664
   buf_next: 087c           buf_prev: 044c           buf_freeLink: 0000
  buf_flags: 04             buf_hVPB: 0279           buf_sector: 0000bcd4
    buf_tid: 00             buf_wrtcnt: 01           buf_wrtcntinc: 0000
   buf_fill: 0000



>> Find the volume these buffers are assigned to.

##ln gdt_vpb
0138:00000098 os2krnl:DOSGDTDATA:GDT_VPB
```

Figure 41. (Part 1 of 2). File System Buffer (BUF)

```
##.d vpb 98:279
        vpb_flink: 01fe                  vpdFAT_cluster_mask: 07
        vpb_blink: 02f4                 vpdFAT_cluster_shift: 03
    vpb_ref_count: 004e                     vpdFAT_first_FAT: 0001
 vpb_search_count: 0000                     vpdFAT_FAT_count: 02
 vpb_first_access: 09                    vpdFAT_root_entries: 0200
    vpb_signature: 444a                  vpdFAT_first_sector: 0000014d
     vpb_flags(2): 02:40                  vpdFAT_max_cluster: 95f5
          vpb_FSC: #0000:ff40                vpdFAT_FAT_size: 0096
           vpi_ID: e2ea4414                vpdFAT_dir_sector: 0000012d
         vpi_pDPB: #04b8:0038                    vpdFAT_media: f8
     vpi_cbSector: 0200                    vpdFAT_next_free: 0000
       vpi_totsec: 0004b0f0                    vpdFAT_free_cnt: 63a8
       vpi_trksec: 003f                  vpdFAT_FATentrysize: 10
        vpi_nhead: 0010                      vpdFAT_IDsector: 00000000
         vpi_pDCS: #0000:0000                  vpdFAT_access: 0000
         vpi_pVCS: #0000:0000                 vpdFAT_accwait: 0000
        vpi_drive: 02                          vpdFAT_pEASFT: #00d0:23da
         vpi_unit: 02
         vpi_text: DOS FAT
         vpi_flags: 0003


>> The file system buffer segment is assigned a unique object owner
>> id.

##.m 0a8:0
*har     par      cpg        va    flg next prev link hash hob   hal
 0003 %feaef04c 00000400 %fe6ef000 001 0002 0023 0000 0000 0003 0000      =0000
 hob   har hobnxt flgs own  hmte  sown,cnt lt st xf
 0003  0003 fec5  0000 ffec 0000  0000 00  01 00 00 vmkrhrw
   pvmli    cs    eip     phlock   cpg      va    flg  hptda hob sig  csig
 %fe82e4c4 002d 0a6800a5 %ac22403c 0001 %fe83c000 0005 024b 0003 ea9f ea9f
##dd %(a8:0)-10 l8
%fe702ff0  0b0c0001 4d5000a2 fe705854 0000cc99
%fe703000  94c4ade4 ff930000 0218005a 0000bc8b
##dd %(a8:0)-10-4+cc98 l8
%fe70fc84  00000000 00000000 ff9300a8 003ec010
%fe70fc94  ffa4000c fe80caac 00020100 ff9e0014
##.mo ff93
ff93 fsbuf
##
```

Figure 42. (Part 2 of 2). File System Buffer (BUF)

Notes:

File system buffers are allocated out of a buffer segment whose selector may be located either from the SAS File System section, offset **+0xa** or from symbol **GDT_BUFFERS**.

The buffer segment contains a header of length **+0x1c**.

Header Offset **+0x0** gives the offset to the head of the list of most recently used buffers.

Header Offset **+0x4** gives the offset to the tail of the list of most recently used buffers.

Each buffer contains a **0x18** byte header followed by **0x200** bytes of data. The buffer header is what is formatted by **.D BUF**.

For a description of the Buffer Header fields, see the File System Buffers in the **System Reference**.

3.4.5.10 BIOS Parameter Block (BPB)

```
##.d bpb bootbp
   BytesPerSector: 0200
SectorsPerCluster: 08
  ReservedSectors: 0001
     NumberOfFATs: 00
      RootEntries: 0200
     TotalSectors: 0000
  MediaDescriptor: f8
    SectorsPerFAT: 00fa
  SectorsPerTrack: 0020
            Heads: 0040
    HiddenSectors: 00000820
  BigTotalSectors: 0007cfe0

##.d bpb minimumbpb
   BytesPerSector: 0200
SectorsPerCluster: 01
  ReservedSectors: 0001
     NumberOfFATs: 00
      RootEntries: 0010
     TotalSectors: 0000
  MediaDescriptor: f0
    SectorsPerFAT: 0001
  SectorsPerTrack: 0009
            Heads: 0001
    HiddenSectors: 00000000
  BigTotalSectors: 003fffff
```

Figure 43. BIOS Parameter Block (BPB)

Notes:

Two system BPBs are locatable at the symbols **BootBPB** and **minimumBPB.**

Others are pointed to from the Device Driver Request Packet for **DosDevIOCtl** command code 2 (build **BPB**).

See 3.4.5.7, “Device Driver (Strategy 1) Request Packet (REQ)” on page 171 for information on formatting Device Driver Request Packets.

For a description of the BPB fields, see the BIOS Parameter Block in the *System Reference*.

3.4.5.11 32-Bit Semaphore Structures (SEM32, OPENQ and MUXQ)

```
##.pb 25
 Slot  Sta BlockID  Name     Type        Addr    Symbol
 0025  blk fe81d2d0 pmshell  Sem32     8001 004b hevLazyWrite

##.d sem32 %fe81d2d0
        Type: Shared Event
       Flags: Reset
       pMuxQ: 00000000
  Post Count: 0000
      pOpenQ: fe56eb10
       pName: fd074e98
 Create Addr: 13f60088

##.pb 2f
 Slot  Sta BlockID  Name     Type        Addr    Symbol
 002f  blk fe86ffdc pmshell

##.d sem32 %fe86ffdc
        Type: Shared Event
       Flags: Reset
       pMuxQ: 00000000
  Post Count: 0000
      pOpenQ: fe56eb02
       pName: NULL (anonymous)
 Create Addr: 12d16b48

##.pb 30
 Slot  Sta BlockID  Name     Type        Addr    Symbol
 0030  blk fe86fe58 pmshell  Sem32     0001 00ce hevSleeper

##.d sem32 %fe86fe58
        Type: Private Event
       Flags: Reset
       pMuxQ: 00000000
  Post Count: 0000
  Open Count: 0001
 Create Addr: 13f62f28
```

Figure 44. Three Types of Event Semaphore

Notes:

32-bit semaphores may be Event or Mutex in type, private or shared in scope and if shared, named or anonymous.

The BlockID of a thread waiting on a 32-bit semaphore is the address of the semaphore structure. The **Type** field in the **.**PB command usually indicates a 32-bit semaphore when in use, however this is not always the case. The next example shows how to determine precisely whether the BlockID points to a 32-bit semaphore.

```
##.pb 33
 Slot  Sta BlockID  Name     Type        Addr    Symbol
 0033  blk fe86fa1c pmshell
##.m %0fe86fa1c

*har     par      cpg        va    flg next prev link hash hob   hal
 0003 %feaef04c 00000400 %fe6ef000 001 0002 0023 0000 0000 0003 0000      =0000
 hob   har hobnxt flgs own  hmte  sown,cnt lt st xf
 0003  0003 fec5  0000 ffec 0000  0000 00  01 00 00 vmkrhrw
   pvmli    cs    eip     phlock   cpg      va    flg  hptda hob sig  csig
 %fe82e380 002d 0a6800a5 %ac22403c 0001 %fe83c000 0005 024b 0003 ea9f ea9f
##dd %0fe86fa1c-10 l8
%fe86fa0c  12d15b4c 54564553 ab97d220 ffc20018
%fe86fa1c  00000010 00000000 c4280001 455000a7
##.mo ffc2
ffc2 semstruc
##.d sem32 %0fe86fa1c
        Type: Private Event
       Flags: Reset
       pMuxQ: 00000000
  Post Count: 0000
  Open Count: 0001
 Create Addr: 00a7c428

##
```

Figure 45. How to Determine Whether a BlockID Points to a 32-Bit Semaphore

Notes:

Except for RAMSEM, MUXWAIT, ChildWait and private conventions the BlockID is an address of a structure or routine that relates to the resource or event being waited for.

The **.M** command is used to identifty the owner of the BlockID. In this case it is the kernel resident heap.

Each resident heap block is prefixed with a 4-byte header. If the low order bit is 0 then the high word of the header contains the owner of the heap block.

32-bit Semaphore structures are allocated from regular resident heap blocks. Thus the owner ID may be seen by displaying storage before the BlockID address.

```
##.pb 56
 0056  blk fe88ad8c mutxwait Sem32     8001 0090 _WINOS2_Settings + 77

##.d sem32 %fe88ad8c
        Type: Shared Mutex
       Flags:
       pMuxQ: fe88ab94
  Request Ct: 0001
       Owner: 0055
Requester Ct: 0001
      pOpenQ: fe5724c2
       pName: fd084368
 Create Addr: 00022e98

##.d openq %fe5724c2

  Pid   Open Count
  ----------------
  00d8     0001
  00d7     0001
  00d6     0001

##da %fd084368
%fd084368 RJM\MUTEX0

##.d muxq %fe88ab94

  pMux
  --------
  fe88aba4

##.d sem32 %fe88aba4
        Type: Shared MuxWait
       Flags: Mutex_Mux
    SR Count: 0003
  SR Pointer: fe88a8f4
  Wait Count: 0001
      pOpenQ: fe571e94
       pName: fd0843c8
 Create Addr: 00022ec4

##da %fd0843c8
%fd0843c8 RJM\MUXWAIT
##.d openq %fe571e94
```

Figure 46. (Part 1 of 2). Mux Wait Semaphores

```
  Pid   Open Count
  ---------------
  00d7     0001
  00d6     0001

##dd %fe88a8f4
%fe88a8f4  00000003 00000004 80000090 00000000
%fe88a904  80000091 00000001 80000094 00000001
%fe88a914  ffbe0014 fe88aba4 00000000 5158554d
%fe88a924  fe88a916 ffc70010 fe88a958 dd2f4580
%fe88a934  000001a2 ffc70010 fe88aa88 dd2f464d
%fe88a944  00000010 ffa4000c fe88a968 000102d5
%fe88a954  ffc70010 fe88a9d8 1bd49ce0 000101a5
%fe88a964  ffa4000c fe88aa7c 0001038d ffc70010

##
```

Figure 47. (Part 1 of 2). Mux Wait Semaphores

Notes:

pOpenQ points to an Open Queue Structure, that list all processes that have access to the 32-bit semaphore. This is formatted using **.**D OPENQ.

pName points to the semaphore name, when no anonymous.

pMuxQ points to a MUXQ structure, that lists any 32-bit MUX wait semaphore address lists that have included this semaphore. In this example we see one MUX list.

The MUX list may be formatted using **.**D SEM32.

Instead of a **pMuxQ**, the MUX semaphore contains a pointer to the semaphore record (**SR Pointer**) and a count of the number of semaphores in the list (**SR Count**).

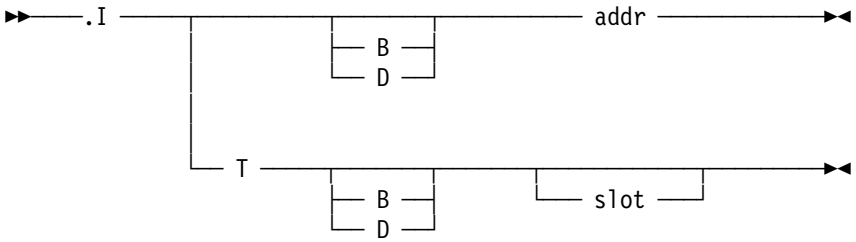
There is no special formatting command for the **SR Structure** - it has to be viewed by displaying storage directly. In this case we see then length, flags and three semaphore handles each followed by the user correlator.

For a description of the 32-bit Semaphore Structures, see the 32-Bit Semaphore Structures in the *System Reference*.

3.4.6 .I - Swap in Storage

Page in a TSD or a Page of Virtual Storage from the Swapper file.

Syntax:



Parameters:

T When specified it requests the Kernel Debugger page in the TSD for a specified thread slot.

One TSD is assigned to each thread slot. If the registers for an out-of-context thread need to be examined, then it may be necessary to swap in the TSD for that slot, since the ring 3 stack frame is stored in the TSD when a thread enters the kernel. The presence or absence of the TSD for a given slot may be deduced by the presence or absence of a value for the **Disp** field of the **.**P command.

If the **T** option is omitted it requests the Kernel Debugger page in the page of virtual storage that encompasses the specified **addr**.

B When specified requests that all breakpoints be re-instated, including those which the current **CS:EIP** may be addressing.

This parameter is effectively obsolete since breakpoints at the current **CS:EIP** are correctly handled by the Breakpoint commands.

D Specifies that a page-in request be scheduled for the kernel debugger daemon thread to execute. In most cases a page-in operation may be performed synchronously, but under the following conditions it is prohibited:

When an interrupt is being handled (that is, not at task time)

When a swapping operation is pending (**TCBfswapping** (**TCB** + 0x1a1) not 0)

When the current thread is blocked (**TK_WF_SLEEPING** (0x40) is set in **TCBWakeFlags** (**TCB** + 0x162))

When in ring0 and InDos is 0

When one of these conditions occurs the page-in request may be scheduled for execution asynchronously by the Debugger Daemon thread by use of the **D** parameter. If the request is successfully scheduled the user is invited to enter the **G** command. The system will dispatch the Daemon thread, in time, which will attempt the page-in request. The Daemon returns control to the debug console using an **INT 3** interrupt.

addr This specifies the virtual address of the page to be paged in. The address is effectively rounded down to the nearest 4K page boundary.

Note: A selector:offset address specification can only be used if the selector does not reference the packed area of the LDT. If it does then a linear address must be supplied by the user.

slot Specifies the thread slot number of the TSD to be paged in. The default slot is the current slot of the debugger's default slot if overridden with the .S command.

Results and Notes:

When an asynchronous page-in is requested, the Kernel Debugger will prompt the user with one of the following depending upon combination of parameters specified:

```
task|addr %nnnn|%nnnnnnnn, LDT entry address %nnnnnnnn queued, G to continue
```

```
task|addr %nnnn|%nnnnnnnn queued, G to continue
```

```
TSD for slot s queued, G to continue
```

On successful completion of a synchronous page-in, the user will be prompted with the command prompt.

If **.I** is unable to complete the request the OS/2 system error code will be displayed (in decimal) in the following message:

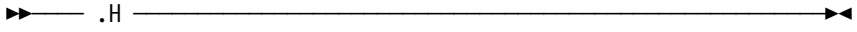
```
OS/2 error code nt
```

Refer to the *Control Programming* reference or to BSEERR.H C header file for an interpretation of the error code.

3.4.7 .H - Display Dump File Header Information

Display dump file header information saved by the stand-alone dump program in the first sector (512 bytes) of the dump file.

Syntax:



Parameters:

None.

Results and Notes:

This command displays the following information:

```
.h
Dump File Header Info:
  Start Addr1: 0
    End Addr1: 2623213
  Total Disks: 9
         Flag: 11
  Ending addresses by disk:
        2623213         6634079         9846551        12950323
       14965147        17345751        19711393
       22092095        25165823
#
```

Each of the fields displayed has the following meaning:

Start Addr1:
The lowest physical address dumped.

End Addr1:
The highest physical address dumped.

Total Disks:
The number of disk volumes the dump data set spans. When the dump is taken to hard disk then the number of volumes is one.

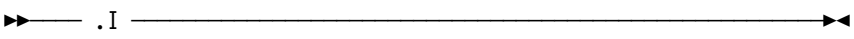
Flag:
Indicates whether the dump file required decompressing. **0** indicates a compressed dump and **11** a decompressed dump.

Ending Addresses by disk
Shows the range of physical memory dumped to each disk volume.

3.4.8 .I (DF) - Show Dump State

Display the dump state.

Syntax:



Parameters:

None.

Results and Notes:

This command displays the following summary information:

```
#.I
PROCESS slot:1c Pid:0003 Ord:0013
PTDA    handle=0088 address=%7bcd5844
MTE     handle=018a address=%fdfadf78 (PMSHL32)
SMTE    address=%fc9a4c48
LDT     handle=0187 address=%7a597000
CODE:   user (cs:eip)#005b:17d679f2 cbargs=
STACKS: user (ss:esp)#0053:01382cbc(active)
        ring2(ss:esp)#09be:00004000(bottom)
        ring0 tcbframe=%7bbd4f54 bottom=%7bbd4f9c
```

Each of the fields display has the following meaning:

slot: The current thread slot at the time the dump was taken. This value is taken from the **TaskNumber** global variable.

Pid: The current Pid when the dump was taken. This value is taken from the **Pid** global variable.

Ord: The Tid of the current thread at the time the dump was taken. This value is taken from the **TCBOrdinal** (**TCB**+ 0x0) of the current **TCB**.

handle= The VMOB handle that represents the control block named to the left.

.I displays this information for the **PTDA, MTE, SMTE and LDT** associated with the current thread when the dump was taken.

See the **.**MO command form more information.

address= The address of the object whose name and handle are given on the same line of display.

user (cs:eip) The current user **CS:EIP** when the dump was taken. See the **.**R command for related information.

cbargs= The call gate argument count if the current task has made a privilege level transition. See the **.**PU command for further information.

user (ss:esp) The current user **SS:ESP** when the dump was taken. See the **.**R command.

ring2(ss:esp) The current ring 2 **SS:ESP** as saved in **TCBCpl2_SS** (**TCB** + 0x1bc) and **TCBCpl2_ESP** (**TCB** + 0x1b8) fields of the current **TCB**.

ring0 tcbframe= The current (or last) kernel entry stack frame pointed to be **TCB_pFrameBase** (**TCB** + 0x3c) when the current thread made a call or transition to the kernel.

bottom= The base of the ring 0 stack (in its all contexts addressable form) for the current thread.

3.4.9 .K - Display User Stack Trace

Display the user stack trace for a given thread slot.

Syntax:

Parameters:

.K Display stack frame trace assuming the default operation size from the descriptor associated with the code selector of the user registers for the specified slot.

.KS Display frame trace assuming an operation size of 16-bits (small model).

.KB Display frame trace assuming an operation size of 32-bits (big model).

slot Display stack trace for thread slot.

The following short hand may be used for the slot number:

***** The current (last) thread the dispatcher gave control to. This value is taken from the word a global label:

```
_TaskNumber
```

\# The debugger default thread slot. This defaults to the current slot unless overridden by the **.**S command.

If no slot number is given then all thread slots are displayed and grouped by process.

Results and Notes:

The **.**K command operates as a K command, but with the starting stack frame and code segment address implicitly determined from the user's register as displayed by the .R command.

The output from the **..**K command display's exactly as the K command but with the slot-number prefixed to the return address when an out-of-context stack trace is displayed. See the following example output.

Attention

The **.**K command is subject to the same limitations as noted for the K command. See the K command description for details.

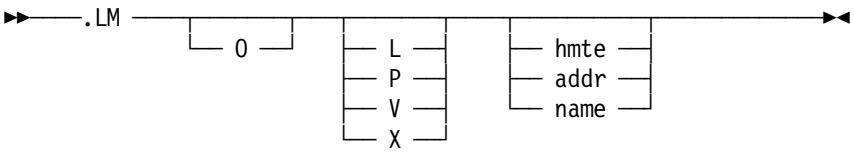
Example output from an out-of-context stack trace:

```
##.S 8
##.K 37
0037|a6e7:0000006f 03d4 0000 00c5 006f
0037|a6e7:00000000 0000 0000 0000 0000
##
```

3.4.10 .LM - Format Loader Structures (MTE, SMTE, OTE and STE)

Display selected information from the MTE and SMTE of one or more loaded modules. Optionally format the associated STE or OTE as follows:

Syntax:



Parameters:

O Format information about each object of each load module.

For 32-bit modules selected fields from the Object Table Entry (OTE) are displayed.

For 16-bit modules selected fields from the Segment Table Entry (STE) are displayed.

L Select Dynamic Link Library modules only. (This includes .FON, .IFS and .DLL modules).

P Select Physical Device Drivers modules only.

V Select Virtual Device Drivers modules only.

X Select Executable modules (.EXE) only.

hmte Specifies the handle of the memory object assigned to the MTE structure to be formatted.

addr Specifies the address of the MTE to be formatted.

name Specifies the name (excluding the file extension and path). The MTE matching this name will be formatted. The name must be specified as a quoted string.

This option requires the SMTE to be present in storage. See below for information on how to make the SMTE present.

The default specification is to scan the entire MTE chain without formatting corresponding STEs or OTEs.

Results and Notes:

The MTE chain is scanned from global symbol:

```
_mte_h
```

When OTE/STE formatting is not requested output appears as follows:

```
.lm
hmte=0293 pmte=%fdfd1a38 mflags=06903140 e:\os2tools\mrfile32.exe
hmte=027f pmte=%fdfd1c80 mflags=06903142 !pulse
hmte=0272 pmte=%fdfd1db4 mflags=06903152 c:\os2\cmd.exe
hmte=00a0 pmte=%fe0177a8 mflags=0698b194 c:\os2\dll\display.dll
hmte=017a pmte=%fe015abc mflags=0698b198 c:\os2\dll\bvhwndw.dll
hmte=010e pmte=%fef282dc mflags=0691b180 ???
hmte=0101 pmte=%fe016b6c mflags=0691b180 ???
hmte=00f9 pmte=%fe016cd4 mflags=0691b180 c:\os2\mdos\vdma.sys
hmte=00f5 pmte=%fe016de0 mflags=0691b180 c:\os2\mdos\vbios.sys
hmte=0072 pmte=%fff2c919 mflags=0002b180 mvdm.dll
hmte=0006 pmte=%fff2bde0 mflags=0000b980 doscalls.dll
hmte=01c8 pmte=%fdf45e78 mflags=0698b1c8 c:\os2\dll\times.fon
hmte=01c6 pmte=%fe017718 mflags=0698b1c8 c:\os2\dll\helv.fon
hmte=00d5 pmte=%fdf32e60 mflags=0608f1ca c:\os2\pmdd.sys
hmte=00d6 pmte=%fdf32f04 mflags=0608f1c9 c:\os2\dos.sys
hmte=00cd pmte=%fdf49f64 mflags=0608f1c9 c:\os2\testcfg.sys
hmte=00cc pmte=%fdf4fb40 mflags=0628a1c9 c:\os2\hpfs.ifs
hmte=00a2 pmte=%fdf45fb8 mflags=0408e1c9 c:\os2dasd.dmd
hmte=00a1 pmte=%fdf32f8c mflags=0408e1c9 c:\ibm2scsi.add
hmte=009f pmte=%fdf2ff18 mflags=0408e1c9 c:\ibm2flpy.add
hmte=0096 pmte=%fdf41f60 mflags=0408e1c9 c:\print02.sys
hmte=0093 pmte=%fdf2efb8 mflags=0408e1c9 c:\clock02.sys
#
```

The fields formatted have the following meaning:

hmte Handle of the memory object occupied by this MTE. Taken from *mte_handle*

pmte Linear address of this MTE

mflags Flag field 1 taken from *mte_flags1*. These flags have the following interpretation:

Table 5 (Page 1 of 2). mflags Interpretation

Name	bit mask	Description
NOAUTODS	0x00000000	No Auto DS exists
SOLO	0x00000001	Auto DS is shared
INSTANCEDS	0x00000002	Auto DS is not shared
INSTLIBINIT	0x00000004	Per-instance Libinit
GINISETUP	0x00000008	Global Init has been setup
NOINTERNFIXUPS	0x00000010	Internal fixups in .EXE-.DLL applied
NOEXTERNFIXUPS	0x00000020	External fixups in .EXE-.DLL applied
CLASS_PROGRAM	0x00000040	Program class
CLASS_GLOBAL	0x00000080	Global class
CLASS_SPECIFIC	0x000000C0	Specific class, as against global
CLASS_ALL	0x00000000	Nonspecific class - all modules
CLASS_MASK	0x00000000	
MTEPROCESSED	0x00000100	MTE being loaded
USED	0x00000200	MTE is referenced

Table 5 (Page 2 of 2). mflags Interpretation		
Name	**bit mask**	**Description**
DOSLIB	0x00000400	Set if DOSCALL1
DOSMOD	0x00000800	Set if DOSCALLS
MTE_MEDIAFIXED	0x00001000	File Media permits discarding
LDRINVALID	0x00002000	Module not loadable
PROGRAMMOD	0x00000000	Program module
DEVDRVMOD	0x00004000	Device driver module
LIBRARYMOD	0x00008000	DLL module
VDDMOD	0x00010000	VDD module
MVDMMOD	0x00020000	Set if VDD Helper MTE (MVDM.DLL)
INGRAPH	0x00040000	In Module Graph
GINIDONE	0x00080000	Global Init has finished
MTEADDRALLOCED	0x00100000	Allocate specific or not
FSDMOD	0x00200000	FSD MTE
FSHMOD	0x00400000	FS helper MTE
MTELONGNAMES	0x00800000	Module supports long-names
MTE_MEDIACONTIG	0x01000000	File Media contiguous memory req
MTE_MEDIA16M	0x02000000	File Media requires mem below 16M
MTEIOPLALLOWED	0x04000000	Module has IOPL privilege
MTEPORTHOLE	0x08000000	Porthole module
MTEMODPROT	0x10000000	Module has shared memory protected
MTENEWMOD	0x20000000	Newly added module
MTEDLLTERM	0x40000000	Gets instance termination
MTESYMLOADED	0x80000000	Set if debugger symbols loaded

name The full path name for the module is displayed to the right of the **mflags** field. The name is taken from the smte_path of the SMTE. If the SMTE is swapped out then the the name is taken from mte_modname (the .DEF file link edit name) and prefixed with an **!** symbol.

Where no path information is given then the module is predefined by the system and does not exist separately as a load module file.

The STE and OTE are displayed when the **O** option is specified. These tables are accessed from the address at SMTE+0x1c. This requires that the SMTE be present in storage. If it is not then the following is returned:

???

Swappable MTE - swapped

To page in the SMTE use **.**LM without parameters to obtain the MTE address from the **pmte** field. The SMTE address is at MTE + 0x4. Use the **.**I command to page in the SMTE storage.

For a 16-bit module the STE is formatted as follows:

```
#.lmo 'hpfs'

hmte=00cc pmte=%fdf4fb40 mflags=0628a1c9 c:\os2\hpfs.ifs
seg  sect psiz vsiz hob  sel  flags
0001 0003 eb24 eb24 0000 0668 8d60 code shr prel rel
0002 0079 d22f d230 0000 0670 8d60 code shr prel rel
0003 00e3 07b5 07b8 0000 0678 8d60 code shr prel rel
0004 00e8 0d8a 0d8c 0000 0680 8d60 code shr prel rel
0005 00f0 0d6e 19c2 0000 0688 8d41 data prel rel
0006 00f7 03fb 03fc 0000 0690 8c41 data prel
0007 00f9 0084 0084 0000 0698 8d41 data prel rel
0008 00fa 0010 0014 0000 06a0 8d41 data prel rel
0009 00fb 0238 0238 0000 06a8 8d41 data prel rel
#
```

The STE fields formatted have the following meaning:

seg Segment number. This is a sequential index of module segments. Index entries appearing in the link-edit map will correspond with these values.

sect (ste_offset) Offset in file to segment data.

psiz (ste_size) File data size.

vsiz (ste_minsiz) Minimum allocation size.

hob (ste_seghdl) Memory object handle of segment data.

sel (ste_selector) Selector assigned to this segment.

flags (ste_flags) Segment type and attribute flags. These interpretations of these are displayed to the right of the flag word. They are assigned as follows:

Table 6 (Page 1 of 2). flags Assignments

Name	bit mask	.lmo msg	Description
STE_CODE	0x0000	code	Code segment type
STE_DATA	0x0001	data	Data segment type
STE_PACKED	0x0002		Segment is packed
STE_SEMAPHORE	0x0004		Segment semaphore
STE_ITERATED	0x0008	iter	Segment data is iterated
STE_WAITING	0x0010	move	Segment is waiting on semaphore
STE_SHARED	0x0020	shr	Segment can be shared
STE_PRELOAD	0x0040	prel	Segment is preload
STE_ERONLY	0x0080	EO	Execute only if code segment
STE_ERONLY	0x0080	RO	Read only if data segment
STE_RELOCINFO	0x0100	rel	If segment has reloc records
STE_CONFORM	0x0200	conf	Segment is conforming
STE_RING_2	0x0800	iopl	Ring 2 selector
STE_RING_3	0x0C00		Ring 3 selector

Table 6 (Page 2 of 2). flags Assignments			
Name	**bit mask**	**.lmo msg**	**Description**
STE_HUGE	0x1000	disc	Huge segment
STE_PAGEABLE	0x2000		Just a page can be faulted in
STE_PRESENT	0x2000		Packed segment already loaded
STE_SELALLOC	0x4000		Used to indicate sel allocated
STE_GDTSEG	0x8000		Used to indicate GTD sel alloc

For a 32-bit module the OTE is formatted as follows:

```
#.lmo 'doscall1'

hmte=00a7 pmte=%fdf59f58 mflags=0698b594 c:\os2\dll\doscall1.dll
obj   vsize    vbase    flags   ipagemap cpagemap hob  sel
0001 00001354 1a010000 80009025 00000001 00000002 00ad d00e r-x shr alias iopl
0002 0000cde8 1a020000 80002025 00000003 0000000d 00ac d017 r-x shr big
0003 00001844 1a030000 80001025 00000010 00000002 00ab d01f r-x shr alias
0004 000002ce 1a040000 80001025 00000012 00000001 00aa d027 r-x shr alias
0005 000054d0 1a050000 8000d025 00000013 00000006 00a9 d02e r-x shr alias conf iopl
0006 00000270 1a060000 80001023 00000019 00000001 00a8 d037 rw- shr alias
0007 00001b40 1a070000 80001003 0000001a 00000002 0000 d03f rw- alias
```

The OTE fields formatted have the following meaning:

obj Object number. This is a sequential index of module object. Index entries appearing in the link-edit map will correspond with these values.

vsize (ote_size) Object virtual size

vbase (ote_base) Object base virtual address

flags (ote_flags) Attribute flags. The interpretations of these are displayed to the right of the each line. They are assigned as follows:

Table 7 (Page 1 of 2). flags Attribues			
Name	**bit mask**	**.lmo msg**	**Description**
OBJREAD	0x00000001	r	Readable Object
OBJWRITE	0x00000002	w	Writeable Object
OBJEXEC	0x00000004	x	Executable Object
OBJRSRC	0x00000008	rsrc	Resource Object
OBJDISCARD	0x00000010	disc	Object is Discardable
OBJSHARED	0x00000020	shr	Object is Shared
OBJPRELOAD	0x00000040	prel	Object has preload pages
OBJINVALID	0x00000080	inv	Object has invalid pages
OBJZEROFIL	0x00000100	zfill	Object has zero-filled pages
OBJRESIDENT	0x00000200		Object is resident
OBJALIAS16	0x00001000	alias	16:16 alias required
OBJBIGDEF	0x00002000	big	Big/Default bit setting

Table 7 (Page 2 of 2). flags Attribues

Name	bit mask	.lmo msg	Description
OBJCONFORM	0x00004000	conf	Object is conforming for code
OBJIOPL	0x00008000	iopl	Object I/O privilege level
OBJMADEPRIV	0x40000000		Object is made private for debug
OBJALLOC	0x80000000		Object allocated and used by loader

ipagemap (ote_pagemap) Object page map index

cpagemap (ote_mapsize) Number of entries in object page map

hob (ote_seghdl) Memory object handle of object data

sel (ote_selector) Selector assigned to this object

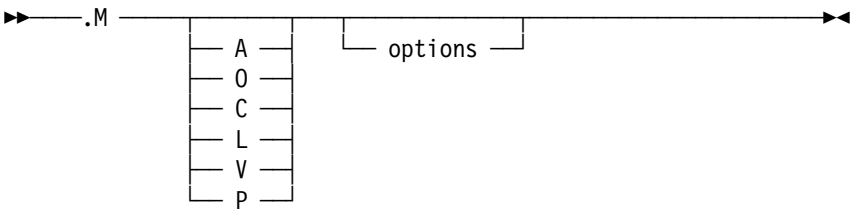
If either the segment table or object is not in storage then the following message is issued:

```
%nnnnnnnnx - swapped
```

3.4.11 .M - Format Memory Structures

Format memory management structures (VMOB, VMAR, VMAL, VMCO, VP and PF).

Syntax:



Parameters:

A Format Memory Arena Records (VMARs). See the **.**MA command for more information.

O Format Memory Object Records (VMOBs). See the **.**MO command for more information.

C Format Memory Context Records (VMCOs). See the **.**MC command for more information.

L Format Memory Alias Records (VMALs). See the **.**ML command for more information.

V Format Virtual Page structures (VPs). See the **.**MV command for more information.

P Format Page Frame structures (PFs). See the **.**MP command for more information.

options See the corresponding **.**M**x** command for details of applicable options.

The **.**M command defaults to:

.MAMC

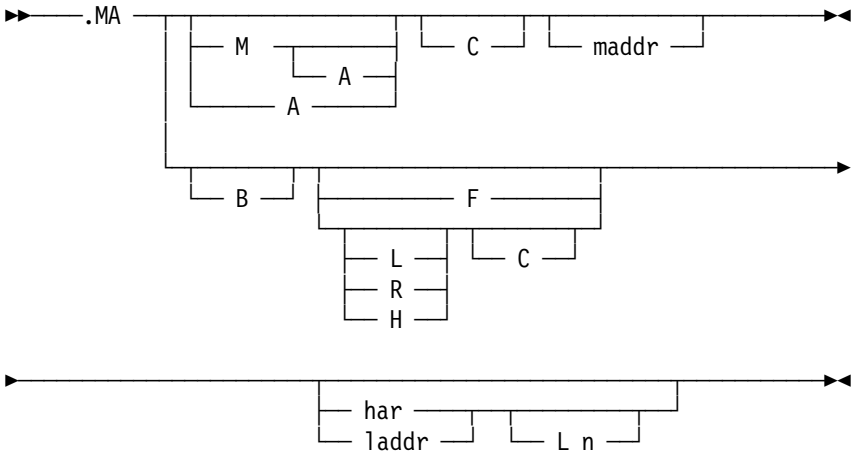
.MAAMC

For further details see the **M** option of the **.**MA command.

3.4.12 .MA - Format Memory Arena Records (VMAR)

Display memory arena records VMARs. Optionally format related object records VMOBs, alias records VMALs and context records VMCOs.

Syntax:



Parameters:

A This option is used with (and implies) the **M** option. It causes a match for private area addresses to be made across all contexts. See the **M** option for further details.

Note:

Under Kernel Debugger the default is to match addresses in the current context only.

Under Dump Formatter address matches are made across all contexts, that is, the **A** option is in permanent effect.

B Display in-use (busy) arena records in sequential order.

C Display chained memory structures.

Chaining causes related memory structures to be displayed in groups, the head of which is indicated by an * suffix. The related structures are:

Aliases to the associated arena record (VMALs).

Arena records of all associated alias records (VMARs).

Shared instance data objects for all related arena records

Context records for shared objects of all associated arena records (VMCOs). See the .MC command.

Object records of all associated arena records (VMOBs). See the .MO command.

F Display free arena records.

H Follow the arena hash chain pointer. The hash chain is used by virtual memory management to look up a memory object for a given context from a linear address.

L Follow the arena forward (left) chain pointer. Arena records for each arena are chained using a double-linked circular chain. The Dump Formatter or Kernel Debugger will not detect wrap-around. This option must therefore be limited by specifying a fixed number of arena records, using the **Ln** operand, or interrupted using Ctrl-C.

M Searches for all arena records (of all contexts) that represent virtual memory that encloses the address specified in **maddr** If **maddr** is not specified then the current **CS:EIP** is taken as the matching address. If the storage is in the private arena Kernel Debugger will search the current context only unless the

An address expression may be specified, via the **A** option. The Dump Formatter always searches for matches in all contexts.

R Follow the arena backward (right) chain pointer. Arena records for each arena are chained using a double-linked circular chain. The Dump Formatter or Kernel Debugger will not detect wrap-around. This option must therefore be limited by specifying a fixed number of arena records, using the **Ln** operand, or interrupted using Ctrl-C.

maddr Specifies the matching address to be used with the **M** option.

laddr Specifies the linear address of a specific arena record to be formatted.

L*n* Specifies the number of arena records to display.

har Specifies the handle of a specific arena record to be formatted.

Results and Notes:

Arena records are in contiguous storage, which is anchored from the address given by global variable:

_parvmOne

Output from the **.**MA command is formatted using a common template with minor variations.

Note: Because a common display template is used for all forms of arena record certain fields will be irrelevant to the records being viewed and may contain garbage information. Specific cases are noted in the examples where this applies.

The following are examples of the nine formats of area record:

Free Arena Record

Sentinel Arena Record

Boundary Sentinel Arena Record

System Arena Records mapped by GDT selectors

System Arena Records not mapped by GDT selectors

Shared Arena Records for shared data

Shared Arena Records for instance data

Private Arena Records for non-shared data

Private Arena Records for shared data

For a description of the fields formatted by **.MA** select **.MA** Output Field Descriptions.

For more examples using of the **.M** family of commands see *Volume I, of the OS/2 Debugging Library 'Exploring Memory Management'* chapter.

3.4.12.1 Free Arena Record

```
har     par      cpg        va    flg next prev link hash hob   hal
0263 %fef2948c 000294a2 %00320000 168 0233 0262 0000 0000 02df 0000      =029e
0264 %fef294a2 000294b8 %00000000 000 0000 0000 0000 0000 0000 0000      =0000
0265 %fef294b8 000294ce %00000000 000 0000 0000 0000 0000 0000 0000      =0000
```

Figure 48. Free Arena Record Display

Notes:

Flag bit 0x001 reset signifies a free record.

The only fields of relevance are **har, par, and cpg**.

Bit positions 0xffe of **flg** and remaining fields may contain garbage from a previous use of the record.

For a description of the fields formatted by **.MA** select **.MA** Output Field Descriptions.

3.4.12.2 Sentinel Arena Record

```
har     par      cpg        va    flg next prev link hash hob   hal
0004 %fef26062 00000000 %60000000 003 0239 0015 0000 0000 ffc0 0000 max=%fffc0000
0005 %fef26078 0000dfb0 %04000000 007 0259 006e 0000 0000 fff0 0000 max=%1fff0000
```

Figure 49. Sentinel Arena Records

Notes:

Flag bit 0x002 set signifies a sentinel record.

hob is not relevant to sentinel records. (The value displayed originates from the **max=** field).

For OS/2 2.1, arena record 4 is sentinel for the system arena.

For a description of the fields formatted by **.MA** select **.MA** Output Field Descriptions.

3.4.12.3 Boundary Sentinel Arena Record

```
har     par      cpg        va    flg next prev link hash hob   hal
0005 %fef26078 0000dfb0 %04000000 007 0259 006e 0000 0000 fff0 0000 max=%1fff0000
```

Figure 50. Boundary Sentinel Arena Record

Notes:

Flag bits 0x006 set signify a boundary sentinel record.

The boundary sentinel indicates the boundary between the shared and private arena address spaces. Consequently there is only one boundary sentinel to be found in a system.

hob is not relevant to sentinel records. (The value displayed originates from the **max=** field).

For OS/2 2.1, arena record 5 is boundary sentinel for the shared arena.

For a description of the fields formatted by **.**MA select **.**MA Output Field Descriptions.

3.4.12.4 System Arena Record Mapped by GDT

```
 har     par      cpg        va    flg next prev link hash hob   hal
 0006 %fef2608e 00000003 %fff20000 009 000f 00b0 0000 0000 0007 0000   sel=0100
 0007 %fef260a4 0000000b %ffe27000 009 0008 001a 0000 0000 0008 0000   sel=0400
 0008 %fef260ba 0000000b %ffe32000 009 0009 0007 0000 0000 0009 0000   sel=0f00
 0009 %fef260d0 00000010 %ffe3d000 009 000b 0008 0000 0000 000a 0000   sel=0120
```

Figure 51. System arena records - Address Space Mapped by a GDT Selector

Notes:

Flag bit 0x008 set signifies a selector mapping.

va value >= that specified in the System Arena Sentinel signifies system area area record.

For a description of the fields formatted by **.**MA select **.**MA Output Field Descriptions.

3.4.12.5 System Arena Record Not Mapped by GDT

```
 har     par      cpg        va    flg next prev link hash hob   hal
 000e %fef2613e 00000001 %fff16000 001 01d9 0083 0000 0000 000f 0000      =0000
 000f %fef26154 00000001 %fff23000 001 0010 0006 0000 0000 0010 0000      =0000
 0010 %fef2616a 00000002 %fff24000 001 0011 000f 0000 0000 0011 0000      =0000

selector
```

Figure 52. System Arena Records - Address Space not Mapped by a GDT

Notes:

Flag bit 0x008 set to 0 signifies a no selector mapping.

va value >= that specified in the System Arena Sentinel signifies system area area record.

For a description of the fields formatted by **.**MA select **.**MA Output Field Descriptions.

3.4.12.6 Shared Arena Record for Shared Data

```
har     par      cpg        va    flg next prev link hash hob   hal
00b2 %fef26f56 00000010 %1a0a0000 379 00b6 00b3 0000 0000 00be 0000  hco=001ed
00b3 %fef26f6c 00000010 %1a090000 3d9 00b2 00a9 0000 0000 00c0 0000  hco=001ee
00b4 %fef26f82 00000010 %1a0e0000 379 00bb 00b5 0000 0000 00c1 0000  hco=0022e
```

Figure 53. Shared Arena, Shared Data

Notes:

Flag bit 0x200 set to 1 signifies shared arena, shared data.

Context records chained from **hco** value will list the processes that currently share the memory object represented by this arena record.

For a description of the fields formatted by **.**MA select **.**MA Output Field Descriptions.

3.4.12.7 Shared Arena Record for Instance Data

```
har     par      cpg        va    flg next prev link hash hob   hal
00e9 %fef27410 00000020 %1abb0000 179 0105 00ea 0000 0000 02b5 0000      =0000
00ea %fef27426 00000010 %1aba0000 179 00e9 00eb 0000 0000 02b6 0000      =0000
```

Figure 54. Shared Arena, Instance Data

Notes:

Flag bit 0x200 set to 0 with a **va** value not in the system arena and 0 hptda indicates shared arena instance data.

Object records chained from **hob** value will list the objects and processes that map to the common virtual address range represented by this arena record.

For a description of the fields formatted by **.**MA select **.**MA Output Field Descriptions.

3.4.12.8 Private Arena Record Non-Shared Data

```
har     par      cpg        va    flg next prev link hash hob   hal
00e7 %fef273e4 00000010 %00020000 179 00e8 00db 0000 0000 011e 0000 hptda=0097
00e8 %fef273fa 00000010 %00030000 179 012f 00e7 0000 0000 011f 0000 hptda=0097
```

Figure 55. Private Non-shared Data, Process Owned Arena Records

Notes:

Arena records not satisfying the criteria for any of the System, Sentinel or Shared Arena records are assumed to be private arena records.

If the private memory object is shared (for example, .EXE code segments running in more than one process) then the associated private arena records for the sharing processes are chained from the **link** field as long as **hal** is zero.

For a description of the fields formatted by **.**MA select **.**MA Output Field Descriptions.

3.4.12.9 Private Arena Record Shared Data

```
har     par      cpg        va    flg next prev link hash hob   hal
01e2 %fef28976 00000010 %00010000 1c9 01e3 01e0 00db 0000 011c 0000 hptda=0237
```

Figure 56. Private Shared Data, Process Owned Arena Records

Notes:

Arena records not satisfying the above criteria are assumed to be private arena records.

If the private memory object is shared (for example, .EXE code segments running in more than one process) then the associated private arena records for the sharing processes are chained from the **link** field as long as **hal** is zero.

For a description of the fields formatted by **.**MA select **.**MA Output Field Descriptions.

3.4.12.10 .MA Output Field Descriptions

Output from **.**MA is in a tabular format of the following form:

```
har     par      cpg        va    flg next prev link hash hob   hal
0005 %fef26078 0000dfb0 %04000000 007 0259 006e 0000 0000 fff0 0000 max=%1fff0000
0009 %fef260d0 00000010 %ffe3d000 009 000b 0008 0000 0000 000a 0000   sel=0120
00b4 %fef26f82 00000010 %1a0e0000 379 00bb 00b5 0000 0000 00c1 0000  hco=0022e
00ea %fef27426 00000010 %1aba0000 179 00e9 00eb 0000 0000 02b6 0000      =0000
00e7 %fef273e4 00000010 %00020000 179 00e8 00db 0000 0000 011e 0000 hptda=0097
```

The field headings have the following meaning:

har The handle of the arena record being formatted. This is the unique identifier by which the arena record is known.

par The linear address of the arena record being displayed.

cpg The size of the contiguous linear address range reserved by this arena record expressed as a hexadecimal number of pages. This value will be greater then or equal to the size of the memory object that occupies this linear address range.

For small allocations (<64K) in non-system arenas this will usually be rounded to 0x10 pages so that the CRMA may be applied.

For free records this field is used as a chain pointer to the next free record but only the low order 20 bits are displayed by **.**MA.

va The linear address of the beginning of the memory object represented by the arena record.

flg Arena record flags. These may take a combination of the following settings:

Name	Bit mask	Description
AR_INUSE	0x001	Record not on free list
AR_TAG	0x006	Record type mask
AR_TAGREG	0x000	Regular record
AR_TAGSEN	0x002	Sentinel
AR_TAGBSEN	0x006	Boundary sentinel
AR_SELMAP	0x008	Memory mapped by selector
AR_SELBASEALL	0x00c	Base selector map all
AR_SELMASK	0x00c	Selector map mask
AR_RELOAD	0x010	Pre-reserved for huge item or or reserved for reload of MTE object
AR_WRITE	0x020	Write permission
AR_USER	0x040	User permission
AR_EXEC	0x080	Executable permission
AR_READ	0x100	Read permission
AR_HCO	0x200	Record linked to Context List
AR_GUARD	0x400	Guard page
AR_SGS	0x800	Registered under Screen Group Switch control.

next Handle of next arena record within the same arena (private, shared or system).

prev Handle of the previous record within the same arena (private, shared or system).

link Handle of an associated arena record.

For private arena sentinel records this points to the boundary sentinel.

For shared arena shared data this points to other private arenas sharing the same object.

For alias objects this points to the arena record of the associated (aliased) object.

hash Handle of the next arena record whose ***va*** hashes to the same hash chain.

hob The handle of the associated memory object record. See **.**MO command.

hal The handle of the associated memory alias record. See **.**ML command. If this field is set to a value of 0xffff then this is not the handle of an alias record, but signifies that this arena has been privatized by the creation of an alias to it.

hptda=*hhhh* The handle of the pseudo-object that is the PTDA of the process that has this arena record assigned to its private address space. Use **.**MO hhhh to display the PTDA pseudo-object and hence obtain the address of its virtual address.

max=*%mmmmmmmm* Maximum linear address of the area headed by this sentinel record.

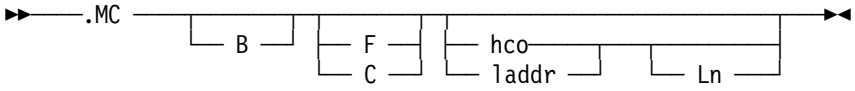
sel=*ssss* GTD selector that is assigned to a system arena memory object.

hco=*ccccc* Handle of the first context record that represents processes sharing this shared arena, shared data memory object. See the **.**MC command.

3.4.13 .MC - Format Memory Context Records (VMCO)

Display memory context records (VMCOs).

Syntax:



Parameters:

B Display in-use (busy) alias records in sequential order.

C Display chained context records.

Chaining causes related context records that are chained from **hconext** of the current context to be formatted. The head of each group indicated by an * suffix. Context records are chained to represent each instance of an object being shared among several processes. The head of the chain is pointed to by the **hco** field of the corresponding arena record See the **.**MA command for information on formatting arena records.

Notes: There is no pointer to the arena record from the context record. Associated arena records have to be found by scanning arena or object records.

The **C** option will not format context records from the head of the chain. Do achieve this, locate the corresponding arena record and use

.MAC har

F Display free alias records.

laddr Specifies the linear address of a specific context record to be formatted.

L*n* Specifies the number of context records to display.

hco Specifies the handle of a specific context record to be formatted.

Results and Notes:

Context records are in contiguous storage, which is anchored from the address given by global variable:

_pcovmOne

Output from the **.**MC command appears in one of two formats:

Free Context Records
Busy Context Records

For a description of the fields formatted by **.MC** select **.MC** Output Field Descriptions.

For more examples using of the **.M** family of commands see: Exploring Memory Management.

3.4.13.1 Free Alias Records

```
hco=00313 pco=ffe4bf7a  pconext=ffe4bf7f
hco=00314 pco=ffe4bf7f  pconext=ffe4bf84
hco=00315 pco=ffe4bf84  pconext=ffe4bf89
hco=00316 pco=ffe4bf89  pconext=ffe4bf8e
```

Figure 57. Free Context Record Display

Notes:

pconext is used as a chain pointer to the next free context record.

For a description of the fields formatted by **.MC** see the **.MC** Output Field Descriptions.

3.4.13.2 Busy Context Records

```
hco=00001 pco=ffe4b020  hconext=00000 hptda=00ae f=1d  pid=0002 c:pmshell.exe
hco=00002 pco=ffe4b025  hconext=00000 hptda=00ae f=1d  pid=0002 c:pmshell.exe
hco=00003 pco=ffe4b02a  hconext=00000 hptda=00ae f=13  pid=0002 c:pmshell.exe
hco=00004 pco=ffe4b02f  hconext=00000 hptda=00ae f=1d  pid=0002 c:pmshell.exe
```

Figure 58. Selector Busy Context Records

Notes:

Flag bit 0x20 set signifies a context handle > 64k. In effect this is a 1 bit extension of the 16 bit **hco** field of the VMCO. The **.ML** command takes this into account when formatting VMCO records.

Flag bit 0x80 set signifies that the context has been privatized. This implies that the object was originally shared but a private instance of it has subsequently been created. Typically this occurs when DosDebug is used to access a debugee's shared data.

For a description of the fields formatted by **.MC** see the **.MC** Output Field Descriptions.

3.4.13.3 .MC Output Field Descriptions

Output from **.MC** appears in one of is of the following forms:

```
hco=00317 pco=ffe4bf8e  pconext=ffe4bf93

hco=00001 pco=ffe4b020  hconext=00000 hptda=00ae f=1d  pid=0002 c:pmshell.exe
```

Each of the fields has the following meaning:

hco= The handle of the context record being formatted. This is the unique identifier by which the context record is known.

pco= The linear address of the context record being formatted.

hconext= For busy records this is the handle of the next context record that represents another user of the related (shared) object. For free records this is the handle of the next free record.

hptda= The handle of the PTDA pseudo-object that represents the process sharing the corresponding memory object.

f Context record flags.

The following flags are defined:

Name	Bit mask	Description
CO_CREATOR	0x01	originating context
CO_PRIV	0x80	Privatized context
CO_HCOH	0x20	Next context record handle > 64K
CO_WRITE	0x02	Write access
CO_USER	0x04	User attribute
CO_EXEC	0x08	Execute access
CO_READ	0x10	Read access
CO_GUARD	0x40	Guard page

Pid= This names the process Id and process executable that corresponds to the **hptda** field.

3.4.14 .MK - Display Memory Lock Information Records (VMLKI)

Display memory lock information records.

Note:

This command was implemented by feature 82818 for the ALLSTRICT kernel only. It is not available in either of the GA versions for OS/2 Warp V3.0 or OS/2 V2.11.

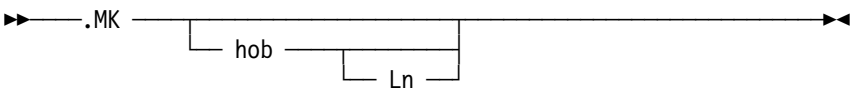
Feature 82818

Feature 82818 introduces the Kernel Debugger **.**MK command. 82818 is supplied as an APAR fix to:

OS/2 Warp V3.0: as PJ18364 in fixpack 7

OS/2 V2.11: as PJ16805 in fixpack 90

Syntax:



Parameters:

none Lists all lock information records for all memory objects with locked records.

L*n* Specifies the number of lock information records to display for a a given **hob**.

hob Specifies the handle of a specific object record whose lock information records are to be formatted.

Results and Notes:

Lock information records are maintained for outstanding memory locks in memory lock information records (**VMLIs**) which are located at the address given by global variable:

_pVMLIHead

When a memory lock request is successfully executed a lock handle is returned to the caller for later use when unlocking memory. The lock handle normally resides in the caller's storage. It comprises a concatenation of:

The requestor's **hptda**

The **hob** whose pages are being locked

The page number

The number of pages

Request flags

In addition a check-sum or signature is calculated from these values and stored with the lock handle.

The **VMLI** is a copy of the constituents of the lock handle that resides in system memory.In addition it includes:

The requestor's return address

A pointer to the next **VMLI**

A pointer to the requestor's lock handle

The **.**MK command formats the contents of the **VMLI** then re-calculates the signature. The calculated and saved signatures should be identical.

Next the lock handle is accessed. If it differs from the corresponding **VMLI** then it too is formatted and the signature is re-calculated and displayed. If either the formatted lock handle and corresponding **VMLI** or the calculated and extracted signatures disagree then a problem may be indicated. For example, an overlayed or freed lock handle. However, there is no requirement for lock requestors to retain their lock handles in their original locations.

Attention

The Kernel Debugger can trap when attempting to format lock handles from freed memory.

Note: When feature 82818 is present **VMLI** records are automatically formatted when displaying memory object records with locked pages, using the **.**MO command.

Output from the **.**MK command appears as follows:

```
##.mk
   pvmli    cs    eip     phlock   cpg      va    flg  hptda hob sig  csig
 %fe679f1c 0170 fffa015d %fd17d480 0001 %013f1000 0003 0091 0424 18aa 18aa
 %fe68a1dc 0170 fffa015d %fd17d468 0001 %013f1000 0003 0091 0424 18aa 18aa
 %fe74539c 0170 fff3e551 %ffe006ff 0002 %fff33000 0001 02f7 0016 0252 0252
 %fe712c54 0170 fff3e551 %ffe00577 0003 %fff38000 0001 02f7 0016 0258 0258
 %fe761e0c 0908 00000878 %7b6b7d0c 0001 %ffee9000 0005 0091 0190 011f 011f
                                   0000 %4c800000 0ff0 0001 0000 0000 d7f5
 %fe777e18 0908 00000841 %7b6b7d0c 0006 %ffeea000 0005 0091 0227 01bc 01bc
                                   0000 %4c800000 0ff0 0001 0000 0000 d7f5
 %fe777e3c 0908 00000809 %7b6b7d0c 0001 %ffef0000 0005 0091 022c 01c2 01c2
                                   0000 %4c800000 0ff0 0001 0000 0000 d7f5
 %fe777e60 0908 0000072b %7c224066 0002 %17c40000 0005 0091 0199 7e72 7e72
 %fe777e84 0908 0000072b %7c224058 0001 %7a022000 0001 0091 0168 a224 a224
 %fe777ef8 0908 000006ee %7c22403c 0001 %7a022000 0001 0091 0168 a224 a224
 %fe777f28 0908 000006ee %7c22404a 0002 %17c80000 0005 0091 0196 7eaf 7eaf
 %fe777f4c 0908 000000a5 %7c22402e 0001 %fe763000 0005 0091 0003 e80c e80c
```

The field headings have the following meaning:

pvmli Address of the **VMLI** record.

cs Code selector of the requestor of the memory locking function. For calls made through a **DevHlp** request this is taken from **TCBpDHRetAddr** (**TCB** + 0x74). For internal requests the immediate caller of **_VmLockMem** is displayed.

A blank value indicates information from the lock handle is being formatted, because it does not agree with the corresponding **VMLI**. See note above.

eip The instruction pointer of the requestor of the memory locking function. For calls made through a **DevHlp** request this is taken from **TCBpDHRetAddr** (**TCB** + 0x74). For internal requests the immediate caller of **_VmLockMem** is displayed.

A blank value indicates information from the lock handle is being formatted, because it does not agree with the corresponding **VMLI**. See note above.

phlock The address of the lock handle buffer supplied by with the lock request.

cpg The number of contiguous pages locked.

va The linear address of the first page locked.

flg The flags saved from the lock request.

The following bit settings are defined:

Bit value	Description
0x01	Lock is a long-term
0x02	Verify lock call
0x04	Lock originated from a DevHlp

hptda The hptda of the lock requestor.

hob The handle of the associated memory object record. See the **.**MO command.

sig The signature value extracted from the **VMLI** or lock handle.

csig The recalculated signature based on information saved in the **VMLI** or lock handle.

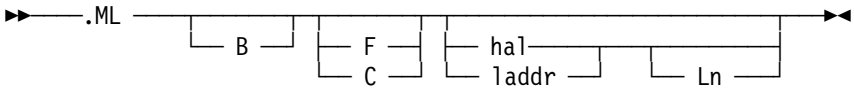
For related information see refer also to the Virtual Memory Lock Trace.

3.4.15 .ML - Format Memory Alias Records (VMAL)

KDB DF (not supported in early 2.1 versions of the Dump Formatter)

Display memory alias records (VMALs). Optionally format related arena records (VMARs), object records (VMOBs) and context records (VMCOs).

Syntax:



Parameters:

B Display in-use (busy) alias records in sequential order.

C Display chained memory structures.

Chaining causes related memory structures to be displayed in groups, the head of which is indicated by an * suffix. The related structures are:

The associated arena record (VMAR). See the **.**MA command.

Aliases to the associated arena record (VMALs).

Arena records of all associated alias records.

Shared instance data objects for all related arena records.

Context records for shared objects of all associated arena records. (VMCOs). See the **.**MC command.

Object records of all associated arena records (VMOBs). See the **.**MO command.

F Display free alias records.

maddr Specifies the matching address to be used with the **M** option.

laddr Specifies the linear address of a specific alias record to be formatted.

L*n* Specifies the number of arena records to display.

hal Specifies the handle of a specific alias record to be formatted.

Results and Notes:

Alias records are in contiguous storage, which is anchored from the address given by global variable:

_palVMAliases

Output from the **.**ML command appears in one of three formats.

Free Alias Record
Selector Alias Record
Linear Address Alias Record

For a description of the fields formatted by **.**ML see the **.**ML Output Field Descriptions.

For more examples using of the **.**M family of commands see Volume I, of the OS/2 Debugging Library ′Exploring Memory Management′ chapter.

3.4.15.1 Free Alias Records

```
hal=000e pal=%ffe61088  palnext=ffe610a0      pgoff=00000  f=000
hal=000f pal=%ffe61090  palnext=ffe61088      pgoff=00000  f=000
hal=0010 pal=%ffe61098  palnext=ffe61090      pgoff=d4460  f=000
```

Figure 59. Free Alias Record Display

Notes:

Flag bit 0x001 reset signifies a free record.

The only fields of relevance are **hal, pal, and palnext**.

For a description of the fields formatted by **.**ML see the **.**ML Output Field Descriptions.

3.4.15.2 Selector Alias Records

```
hal=000a pal=%ffe61068  har=0208  cs=0056 ds=d446 cref=001 f=13
hal=000b pal=%ffe61070  har=020b  cs=0056 ds=d446 cref=001 f=13
```

Figure 60. Selector Alias Record Display

Notes:

Flag bit 0x02 set signifies a CS alias record.

Flag bit 0x10 set signifies that DS is valid, i.e CS is an alias for the same storage mapped by DS. For example after use of DosCreateCSAlias. This flags distinguishes this form of the alias record.

This alias applies to selectors within a specific (process) context.

For a description of the fields formatted by **.**ML see the **.**ML Output Field Descriptions.

3.4.15.3 Linear Address Alias Records

```
hal=0001 pal=%ffe61020  har=00b8  hptda=009f  pgoff=00000  f=001
hal=0002 pal=%ffe61028  har=00b9  hptda=009f  pgoff=00000  f=001
hal=0003 pal=%ffe61030  har=001b  hptda=009f  pgoff=00000  f=001
```

Figure 61. Linear Address Alias Record Display

Notes:

Flag bit 0x10 reset distinguishes this form of the alias record.

hptda and har refer to the context and arena which has been aliases.

The context and arena of the creator of the alias may be shown using

.MLC hal

For a description of the fields formatted by **.**ML see the **.**ML Output Field Descriptions.

3.4.15.4 .ML Output Field Descriptions

Output from **.**ML appears in one of is of the following forms:

```
hal=0006 pal=%ffe61048  har=01b8  hptda=009f  pgoff=00000  f=021

hal=000a pal=%ffe61068  har=0208  cs=0056 ds=d446 cref=001 f=13

hal=0011 pal=%ffe610a0  palnext=ffe610a8      pgoff=00000  f=000
```

Each of the fields has the following meaning:

hal The handle of the alias record being formatted. This is the unique identifier by which the alias record is known.

har The handle of the arena record which represents the aliasing linear address range. The arena record for the aliased linear address is pointed to by the **link** field of the aliasing **har**.

hptda The handle of the PTDA object of the context which has been aliased. This may also take the value of a system owner when memory is in the process of being freed.

pgoff The page offset into the arena which is aliased. Note: aliases may map partial objects. The number of pages aliased may be determined from the arena record which represents the aliased memory. Use **.**MLC hal to display this information.

cs The Code Selector created to alias R/W memory.

ds The Data Selector which has been aliased by a Code selector.

cref The reference count for the number of time a Code Selector alias has been requested for a given Data Selector.

palnext The pointer to the next free alias record.

f Alias record flags. For Selector Aliases this is a 6-bit field, for other aliases this is a 12-bit field.

The following flags are defined:

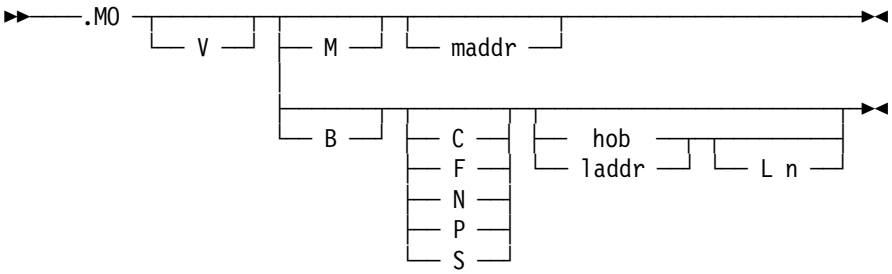
Name	Bit mask	Description
AL_ISBUSY	0x1	Set if record is busy
AL_CSALIAS	0x2	Set is CS alias record
AL_MEMMAP	0x4	Set if MemMapAlias Record
AL_DBGALIAS	0x8	Set if Debug alias
AL_CSDSVALID	0x10	Set if DS selector valid
AL_DEVHLP	0x20	Set if Devhlp Alias
AL_PRIV	0x40	Set if privatized alias
AL_VDM	0x80	Set if VMD alias
AL_NOALIAS	0x100	Set if UVIRT mapping in VDMs

3.4.16 .MO - Format Memory Object Records (VMOB)

Display memory object records (VMOBs). Optionally format related alias records (VMALs), arena records (VMARs) and context records (VMCOs).

If feature 82818 is installed, then under the Kernel Debugger only, lock information records will be formatted whenever a memory object with locked pages is displayed. See the .MK command for more information.

Syntax:



Parameters:

B Display in-use (busy) object records in sequential order.

C Display chained memory structures.

Chaining causes related memory structures to be displayed in groups, the head of which is indicated by an * suffix. The related structures are:

The associated arena record (VMAR). See the **.**MA command.

Aliases to the associated arena record (VMALs).

Arena records of all associated alias records.

Shared instance data objects for all related arena records.

Context records for shared objects of all associated arena records (VMCOs). See the **.**MC command.

Object records of all associated arena records (VMOBs).

Note: Pseudo-objects have no related memory objects.

F Display free object records.

M Searches for a pseudo-object whose address matches the address specified for *maddr*. If no match is found then nothing is displayed.

Notes: The pseudo-object address specified must be an exact match for hit.

The pseudo-object address is that of the object itself and not the VMOB that represents it.

A ***selector:offset*** form of address may not be specified. However a physical address may be specified in order to bypass virtual address validation done by Kernel Debugger and Dump Formatter.

N Specifies that only normal object records be displayed. These are objects whose linear address allocation is represented by an arena record. Contrast this with Pseudo-Object and System Object. See also the **.**MA command for details of arena record display.

P Specifies that pseudo-object records be displayed.

S Specifies that objects to be displayed are those whose memory management semaphore is busy or wanted. The memory management semaphore is used internally for serializing access to memory management structures. It should not be confused with the memory locking as provided by the DevHlp_Lock, DevHlp_Unlock, DevHlp_VMLocl and DevHlp_VMUnlock calls.

V Specifies verbose mode of display. The address of the VMOB structure is displayed but object description and owner interpretation is suppressed.

maddr Specifies the matching address to be used with the **M** option.

An address expression may be specified.

laddr Specifies the linear address of a specific object record to be formatted.

An address expression may be specified.

L*n* Specifies the number of object records to display.

hob Specifies the handle of a specific object record to be formatted according to the criteria specified by the other options.

Results and Notes:

Object records are located in contiguous storage, which is anchored from the address given by global variable:

_pobvm0ne

Output from the **.**M0 command appears in one of four formats:

Normal Object
Pseudo-Object
Free Object Record
System Object

For a description of the fields formatted by **.MO** see the **.MO** Output Field Descriptions.

For more examples using of the **.M** family of commands see Volume I, of the OS/2 Debugging Library 'Exploring Memory Management' chapter.

3.4.16.1 Normal Object Records

```
hob   har hobnxt flgs own  hmte  sown,cnt lt st xf
000d  000c 0000  0325 ffba 0000  0000 00  00 00 00 lock
000e  000d 0000  0000 ffaa 0006  0000 00  00 00 00 os2krnl
000f  000e 0000  0000 ffaa 0006  0000 00  00 00 00 os2krnl
0010  008f 0000  402c 00ae 0115  0000 00  00 00 00 priv 0002 c:pmshell.exe
0011  0010 0000  0000 ff37 0000  0000 00  00 00 00 romdata
```

Figure 62. Normal Object Record Display

```
hob     pob     har hobnxt flgs own  hmte  sown,cnt lt st xf
0001 %fec80020  0001 fec8  0000 fff1 0000  0000 00  00 00 00
0002 %fec80030  0002 fec8  0000 ffe3 0000  0000 00  00 00 00
0003 %fec80040  0003 fec8  0000 ffec 0000  0000 00  01 00 00
```

Figure 63. Normal Object Record Display - Verbose Form

Notes:

The verbose form is specified using the **V**option. This causes the suppression of **owner** and **hmte** interpretation.

For a description of the fields formatted by **.MO** see the **.MO** Output Field Descriptions.

3.4.16.2 Pseudo-Object Records

```
hob       va     flgs own  hmte  sown,cnt lt st xf
0004  %fff13238  8000 ffe1 0000  0000 00  00 00 00 vmah
0005  %fff13190  8000 ffe1 0000  0000 00  00 00 00 vmah
0006  %fff0a891  8000 ffa6 0000  0000 00  00 00 00 mte       doscalls.dll
0072  %ffe3c7d4  8000 ffcb 0000  0000 00  00 00 00 ptda 0001 *sysinit
007a  %fff0b3fa  8000 ffa6 0000  0000 00  00 00 00 mte       mvdm.dll
007b  %fff0b26b  8000 ffa6 0000  0000 00  00 00 00 mte       fshelper.dll
0091  %fe7349ac  8000 ffa6 0000  0000 00  00 00 00 mte       c:pmshapim.dll
0098  %7b9e4060  8000 ffe1 0000  0000 00  00 00 00 vmah
009d  %fe722fb8  8000 ffa6 0000  0000 00  00 00 00 mte       c:clock02.sys
```

Figure 64. Pseudo-Object Record Display

Notes:

The 0x8000 flag bit signifies as psuedo-object.

For a description of the fields formatted by **.MO** see the **.MO** Output Field Descriptions.

3.4.16.3 Free Object Records

```
hob       va     flgs own  hmte  sown,cnt lt st xf
02b5  %fec82b70  0000 0000 0000  0000 00  00 00 00 free
02b6  %fec82b80  0000 0000 0000  0000 00  00 00 00 free
02b7  %fec82b90  0000 0000 0000  0000 00  00 00 00 free
02b8  %fec82ba0  0000 0000 0000  0000 00  00 00 00 free
02b9  %fec82bb0  0000 0000 0000  0000 00  00 00 00 free
```

Figure 65. Free Object Record Display

Notes:

Flag bit 0x001 reset signifies a free record.

The only fields of relevance are **va** and **pob** when the **V** option is specified.

The **va** field is used a link field to other free VMOBs.

For a description of the fields formatted by **.**MO see the **.**MO Output Field Descriptions.

3.4.16.4 System Object IDs

```
fff0 vmllock
fff1 vmob
fff2 vmsgs
fff3 vmbmp16
```

Figure 66. Example System Object Display

Notes:

System object IDs are not represented by VMOB structures. They are pre-defined IDs for system components.

The Dump Formatter and Kernel Debugger display only object names when displaying system objects.

3.4.16.5 .MO Output Field Descriptions

Output from **.**MO appears tabular for with one of the following headings:

```
hob       va     flgs own  hmte  sown,cnt lt st xf
0004  %fff13238  8000 ffe1 0000  0000 00  00 00 00 vmah
0006  %fff0a891  8000 ffa6 0000  0000 00  00 00 00 mte       doscalls.dll

hob     pob     har hobnxt flgs own  hmte  sown,cnt lt st xf
0003 %fec80040  0003 fec8  0000 ffec 0000  0000 00  01 00 00
0004 %fec80050  %fff13238  8000 ffe1 0000  0000 00  00 00 00

hob   har hobnxt flgs own  hmte  sown,cnt lt st xf
0001  0001 fec8  0000 fff1 0000  0000 00  00 00 00 vmob
0002  0002 fec8  0000 ffe3 0000  0000 00  00 00 00 vmar
```

Each of the heading fields has the following meaning:

hob The handle of the object record being formatted. This is the unique identifier by which the object record is known.

hobnext The handle of next shared-instance data object that maps to the same linear address range (shares the same arena record but maps to a different physical address).

har The handle of the arena record that describes the linear address range allocated to this object.

pob The linear address of the object record.

va The virtual address of the pseudo-object named in the object description.

flgs Object record flags.

The following flags are defined:

Name	Bit mask	Description
OB_PSEUDO	0x8000	Pseudo-object
OB_API	0x4000	API located object
OB_LOCKWAIT	0x2000	Waiting for a lock conflict to resolve
OB_LALIAS	0x1000	Object has aliases
OB_SHARED	0x0800	Object's contents are shared
OB_UVIRT	0x0400	UVirt object
OB_ZEROINIT	0x0200	Object is zero-initialized
OB_RESIDENT	0x0100	Initial allocation was resident
OB_LOWMEM	0x0040	Object is in low memory
OB_GUARD	0x0080	Guard page attribute
OB_EXEC	0x0020	Executable attribute
OB_READ	0x0010	Readable attribute
OB_USER	0x0008	User attribute
OB_WRITE	0x0004	Writeable attribute
OB_HUGE	0x0002	Object is huge
OB_SHRINKABLE	0x0001	Object is Shrinkable (only if also OB_SHARED)
OB_DHSETMEM	0x0001	DevHlp_VMSetMems are allowed the object

Notes: See Pseudo-Objects when OB_PSEUDO is set.

OB_API is set as a result of allocation made by some APIs (for example, DosExecPgm). It forces page alignment and signals a likelihood of long-term locking.

OB_HUGE is set when the object is created by DosAllocHuge API.

When OB_LOCKWAIT is set then the thread has detected a lock request conflict and wishes to wait for the conflict to resolve. The conflict occurs because a contiguous storage lock has been requested but cannot be satisfied because one or more of the pages are already short-term locked. If the current request is for a short-term lock then the thread will

wait up to 10 seconds for the conflict to clear. If the time-out expires then the current short-term lock request ends in error and the following message appears on the debugger screen:

```
VMLOCK: Short term lock for > 10 secs: hob=hob
```

If the current request is for a long-term lock then the thread will wait indefinitely. In both cases the blockID the thread waits on is the address of the VMOB flag word (VMOB+0x4). See **.**PB command for information on thread slots waiting on BlockIDs.

own This is the **hob** of the owner of this object. The owning **hob** may be in one of three categories:

1. **System Owner** Used to indicate system owned objects. The owner description usually indicates the type of object that is being displayed. For example, the LDT for process 9 running pulse.exe is owned by system object **0xffb9** and has a description

   ldt 0009 c:pulse.exe.

2. **Module Owner (hmte)** This is used for objects that are part of a load module. The **.hmte** of the load module is used as a the *Owner Id* for the object. The object description names the owning module from the MTE/SMTE structures. See the **.**LM command for related information.

3. **Process owner (hptda)** Process owned objects are those allocated in the private or shared arenas under a running process and not part of a loaded module. The **hptda** of the process is used as the owner. The owner description names the process id and main executable module.

hmte This names the hmte or System Object Id of the executable code that allocated the memory object.

sown Semaphore owner id. This is the thread slot number that owns the memory management semaphore associated with this object. Memory management uses the address of the VMOB as the BlockID to sleep on when the semaphore is held. This semaphore is used to serialize access to a VMOB structure. See the **.**PB command for information on thread slots waiting on BlockIDs.

cnt Count of owners of the VMOB semaphore and wait flag.

The low order bit of **cnt** is used as a wait indicator. The high order 7 bits are a count of the number of times the owning thread has requested the VMOB semaphore without releasing it. See **sown** filed above for related information.

xf Extra flags.

The following flags are defined:

Name	Bit mask	Description
VMOB_SLOCK_WAIT	0x01	Waiting on short term locks to clear
VMOB_LLOCK_WAIT	0x02	Waiting on long term locks to clear
VMOB_DISC_SEG	0x04	Object is part of a discardable seg

Name	Bit mask	Description
VMOB_HIGHMEM	0x08	Object was allocated via dh_vmalloc using the VMDHA_USEHIGHMEM flag

Notes:

The lock wait flags indicate that a thread is waiting for locked pages in the memory object to be unlocked, but not to resolve a conflicting lock request: that is indicated with the **OB_LOCKWAIT** flag.

If a thread blocks waiting for long-term locks to clear then the address of the long-term lock count (VMOB + 0xd) is used as the BlockID the thread blocks on. The thread blocks indefinitely.

If a thread blocks waiting for short-term locks to clear then the address of the short-term lock count (VMOB + 0xe) is used as the BlockID the thread blocks on. The thread will block for up to 10 seconds. If after that time the short-term lock has not been cleared then an error is returned and under the debug kernel the following message is sent to the debug console:

VMLOCK: Short term lock for > 10 secs: hob=**hob**

See the 3.4.21, ".PB - Display Blocked Thread Information" on page 246 (.PB command) for information on thread slots waiting on BlockIDs.

lt Count of active long-term lock holders. A non-zero value indicates one or more pages of the memory object have been long-term locked, that is prevented from being paged out from physical storage. Long-term locks are expected to be held for a relatively long period of time, in the order of seconds. See the **.**MP command for information on displaying physical storage status. See also VM Lock Trace Kernel Debugger facility.

st Count of active short-term lock holders. A non-zero value indicates one or more pages of the memory object have been short-term locked, that is prevented from being paged out from physical storage. Short-term locks are expected to be held for a relatively short period of time, in the order of milliseconds. See the **.**MP command for information on displaying physical storage status. See also 1.4.2, "Virtual Memory Management Lock Trace" on page 18 Kernel Debugger facility.

description The object description appears to the right of the tabular display. It is combines the interpretation of **own** and **hmte** fields. The following forms are possible:

Process owned objects

These appear as:
priv *Pid process*
where:

Pid Is the owning process id
process Is the main executable running the owning process

MTE Owned objects

These appear as:
shared *module*
where:

module Is the name of the module that contains the object (**hob**) displayed.

PTDA Pseudo-objects

These appear as:
ptda *Pid process*
where:

Pid Is the process id in which the object is located.
process Is the main executable running the owning process

MTE Pseudo-objects

These appear as:
mte *module*
where:

module Is the module name that corresponds to the MTE pointed to by the **va**

LDT occupying storage

This appears as:
ldt *Pid process*
where:

Pid Is the id of the process that owns the LDT related to the object
process Is the main executable running the owning process

Free objects

These appear as:
free

System Owned Objects

These appear as:
owner user
where:

owner Is the system object name corresponding to the **own** field.
user Is the system object name corresponding to the **hmte** field.

System Object Owner IDs: *System objects* are a reserved range of **hobs** used to attribute ownership of virtual memory objects to system components. System object IDs have no corresponding VMOB.

The following table lists the system objects IDs are defined. The names shown are those displayed by the Kernel Debugger and Dump Formatter when formatting VMOB structures:

Table 8 (Page 1 of 6). System Object IDs		
Name	**ID**	**Description**
lielist	0xff2d	LDR LieLists
demversion	0xff2e	DEM fake version entries
vmbmapd	0xff2f	VM Arena Bitmap Directory
npipenpn	0xff30	Named pipe NPN segment
npipenp	0xff31	Named pipe NP segment
reqpkttcb	0xff32	DD TCB request packets
reqpkt2	0xff33	DD strat2 request packets
spldevrmp	0xff34	Spool Dev RMP segment
chardevrmp	0xff35	Char Dev RMP segment
syssemrmp	0xff36	System Semaphore RMP segment
romdata	0xff37	ROM data
libpath	0xff38	LDR LibPath
jfnflags	0xff39	JFN flags
jfntable	0xff3a	JFN table
ptouvirt	0xff3b	PhysToUVirt
tkr3stack	0xff3c	Ring 3 stack
tkr2stack	0xff3d	Ring 2 stack
tkenv	0xff3e	User Environment
tktib	0xff3f	Thread Information Block
reqpkt1	0xff40	DD strat1 request packets
allocphys	0xff41	Allocated via DevHlp AllocPhys
khbdon	0xff42	Unusable donated heap page owner
krhrw1m	0xff43	Resident R/W 1Meg mem heap owner
krhro1m	0xff44	Resident R/W 1Meg mem heap owner
mmph	0xff45	dekko mapped memory
pageio	0xff46	pageio per-swap-file save block
fsreclok	0xff47	Record lock record owner
		File System Drivers
fsd1	0xff48	FSD 1
fsd2	0xff49	FSD 2
fsd3	0xff4a	FSD 3
fsd4	0xff4b	FSD 4
fsd5	0xff4c	FSD 5
fsd6	0xff4d	FSD 6
fsd7	0xff4e	FSD 7
fsd8	0xff4f	FSD 8 and subsequent
		Device Drivers
dd1	0xff50	Device driver 1
dd2	0xff51	Device driver 2
dd3	0xff52	Device driver 3

Table 8 (Page 2 of 6). System Object IDs

Name	ID	Description
dd4	0xff53	Device driver 4
dd5	0xff54	Device driver 5
dd6	0xff55	Device driver 6
dd7	0xff56	Device driver 7
dd8	0xff57	Device driver 8
dd9	0xff58	Device driver 9
dd10	0xff59	Device driver 10
dd11	0xff5a	Device driver 11
dd12	0xff5b	Device driver 12
dd13	0xff5c	Device driver 13
dd14	0xff5d	Device driver 14
dd15	0xff5e	Device driver 15
dd16	0xff5f	Device driver 16 and subsequent
		Miscellaneous Owners
fsclmap	0xff60	Cluster map owner
cdsrmp	0xff61	Current Directory Structure RMP seg
tom	0xff62	Timeout Manager
abios	0xff63	Advanced BIOS
cache	0xff64	Cache
dbgdcb	0xff65	DBG Debug Control Block
dbgkdb	0xff66	DBG Kernel Debug Block
dbgwpcb	0xff67	DBP Watch Point Control Block
demsft	0xff68	DEM SFT array (for FCBs)
demfonto	0xff69	DEM font offsets
demfont	0xff6a	DEM font data
devhlp	0xff6b	Allocated via devhlp AllocPhys
discard	0xff6c	Discardable, zero fill object
doshlp	0xff6d	DosHelp segment
dyndtgp	0xff6e	DYN trace point parm block
dyndto	0xff6f	Dynamic trace point
dyndtot	0xff70	Tmp dynamic trace info
dynmtel	0xff71	DYN MTE dynamic trace link
emalloc	0xff72	EM86 malloc()
emtss	0xff73	EM86 TSS
device	0xff74	Installed device driver
infoseg	0xff75	infoseg (local or global)
initmsg	0xff76	INIT saved message
init	0xff77	Generic init-time only
intdirq	0xff78	INT IRQ info
intstack	0xff79	Interrupt stack

Table 8 (Page 3 of 6). System Object IDs		
Name	**ID**	**Description**
iopllist	0xff7a	List of modules with IOPL
kdbalias	0xff7b	Kernel debugger alias
kdbsym	0xff7c	Kernel debugger symbol
kmhook	0xff7d	KM hook info
ksem	0xff7e	KSEM semaphore
lbdd	0xff7f	Loadable base device driver
lid	0xff80	ABIOS logical identifier
monitor	0xff81	Monitor segment
mshare	0xff82	Named-shared
mshrmp	0xff83	RMP having mshare records
nmi	0xff84	Non maskable interrupt
npx	0xff85	287/387 save area
orphan	0xff86	Orphaned segment
prof	0xff87	Profile support
ptogdt	0xff88	Allocated via dh_allocateGDTSelector
ptovirt	0xff89	PhysToVirt
puse	0xff8a	Page Usage
pusetmp	0xff8b	Tmp Page Usage
perfview	0xff8c	Perfview
qscache	0xff8d	QuerySysInfo cache
ras	0xff8e	RAS segment
resource	0xff8f	Resource BMP segment
sysserv	0xff90	System service
timer	0xff91	Timer services segment
traphe	0xff92	TRAP Hard Error
		File System Owners
fsbuf	0xff93	File system buffer
cdevtmp	0xff94	Char DEV TMP
fsc	0xff95	FSC segment
dpb	0xff96	DPB
eatmp	0xff97	FAT EA TMP
fatsrch	0xff98	FAT search segment
gnotify	0xff99	FindNotify global segment
pnotify	0xff9a	FindNotify private segment
fsh	0xff9b	Installable file sys helper
ifs	0xff9c	Installable file system
mfsd	0xff9d	Mini file system
mft	0xff9e	Master file table
npipebuf	0xff9f	Named pipe I/O buffer segment
pipe	0xffa0	Pipe

Table 8 (Page 4 of 6). System Object IDs

Name	ID	Description
sft	0xffa1	System file table
vpb	0xffa2	Volume parameter block
		Loader Owners
ldcache	0xffa3	Loader Instance Data Cache
ldrdld	0xffa4	LDR Dynamic Load record
invalid	0xffa5	Cache being made
ldrmte	0xffa6	mte
ldrpath	0xffa7	LDR MTE path
ldrnres	0xffa8	LDR non-resident names
prot16	0xffa9	Protect 16 list
		Boot Loader and Kernel Owners
os2krnl	0xffaa	OS2KRNL load image
os2ldr	0xffab	OS2LDR load image
ripl	0xffac	Remote IPL (remote boot)
		Page Manager Owners
pgalias	0xffad	Temporary page manager aliases
pgbuf	0xffae	PG loader and swapper buffer
pgcrpte	0xffaf	PG Compat. region page table
dbgalias	0xffb0	Debugger alias pte
pgdir	0xffb1	PG Page directory
pgkstack	0xffb2	Kernel stack region
pgvp	0xffb3	VP array
pgpf	0xffb4	PF array
pgprt	0xffb5	Page Range Table
pgsyspte	0xffb6	PG System page tables
		Selector Manager Owners
gdt	0xffb7	SEL GDT
selheap	0xffb8	Selector-mapped heap block
ldt	0xffb9	SEL LDT
lock	0xffba	SEL Lock
selnop	0xffbb	NO-OP Locks
seluvirt	0xffbc	SEL UVIRT mapping
		Semaphore Owners
semmisc	0xffbd	SEM Miscellaneous
semmuxq	0xffbe	SEM Mux Queue
semopenq	0xffbf	SEM Open Queue
semrec	0xffc0	SEM SemRecord
semstr	0xffc1	SEM string
semstruc	0xffc2	SEM Main structure
semtable	0xffc3	SEM Private/Shared table

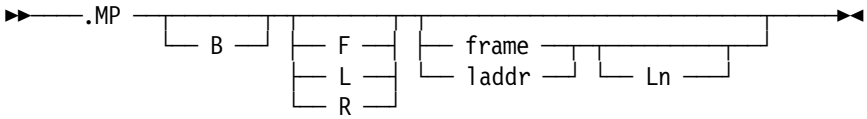
Table 8 (Page 5 of 6). System Object IDs		
Name	**ID**	**Description**
		Swapper Owners
smdfh	0xffc4	SM Disk Frame Heap
smsfn	0xffc5	SM SFN array
smsf	0xffc6	SM Swap Frame
		Tasking Owners
tkextlst	0xffc7	TK Exit List record
tkkmreg	0xffc8	TK dispatch (KM) registers
tklibif	0xffc9	TK LibInit Free Notification record
tklibi	0xffca	TK LibInit record
ptda	0xffcb	TK PTDA
tcb	0xffcc	TK TCB
tsd	0xffcd	TK TSD
		VDD, VDH, VDM Owners
vddblkh	0xffce	VDD block header
vddblk	0xffcf	VDD memory block
vddcfstr	0xffd0	VDD config.sys string
vddctmp	0xffd1	VDD creation tmp allocation
vddep	0xffd2	VDD Entry Point
vddheaph	0xffd3	VDD heap header
vddheap	0xffd4	Heap objects to load VDDs
vddhook	0xffd5	VDD hook
vddla	0xffd6	VDD Linear Arena header
vddlr	0xffd7	VDD Linear arena Record
vddmod	0xffd8	VDD module record
vddopen	0xffd9	Open VDD record
vddpddep	0xffda	VDD PDD Entry Point
vddproc	0xffdb	VDD procedure record
vddstr	0xffdc	VDD string
vdhfhook	0xffdd	VDH fault hook
vdhalloc	0xffde	VDH services resident memory
vdhswap	0xffdf	VDH services swappable memory
vdmalias	0xffe0	VDM Alias
		Virtual Memory Manager Owners
vmah	0xffe1	VM arena header
vmal	0xffe2	VM Alias Record
vmar	0xffe3	VM Arena Record
vmbmap	0xffe4	VM Location Bitmap
vmco	0xffe5	VM Context Record
vmdead	0xffe6	VM Dead Object
vmhsh	0xffe7	VM Location Hash Table

Table 8 (Page 6 of 6). System Object IDs		
Name	**ID**	**Description**
vmkrhb	0xffe8	VM *UNKNOWN* busy KRHB
vmkrhf	0xffe9	VM free KRHB
vmkrhl	0xffea	VM end KRHB
vmkrhro	0xffeb	VM Public Kernel Resident R/O Heap
vmkrhrw	0xffec	VM Public Kernel Resident R/W Heap
vmkshd	0xffed	VM Swappable Heap Descriptor
vmkshro	0xffee	VM Public Kernel Swappable R/O Heap
vmkshrw	0xffef	VM Public Kernel Swappable R/W Heap
vmllock	0xfff0	VM long term lock manager
vmob	0xfff1	VM Object Record
vmsgs	0xfff2	VM Screen Group Switch record
vmbmp16	0xfff3	VM Temp buf (BMP16)
shrind	0xfff4	Reserved for shared indicator
give	0xfff5	Giveable segment
get	0xfff6	Gettable segment
giveget	0xfff7	Giveable and gettable segment
preload	0xfff8	Loader's preload object

3.4.17 .MP - Format Memory Page Frame Structures (PFs)

Display memory Page Frame Structures (PFs).

Syntax:



Parameters:

B Display in-use (busy) Page Frame Structures in sequential order.

Note: In-Use PFs are signified by the PF_FREE flag being reset and not by the PF_BUSY flag being set.

F Display free Page Frame Structures.

L Follow left (forward) chain pointer. This is only of relevance to Free and Idle Page Frame Structures since these are linked in a double

linked chain. **Warning:** Both the Dump Formatter and the Kernel Debugger may fail to recognize the chain pointers correctly. In particular the 2 high order digits of the frame number are truncated. Use this option advisedly! See the Free Page Frame Structures or Idle Page Frame Structures for information on locating Idle and Free PF chains.

R Follow right (backward) chain pointer. This is only of relevance to Free and Idle Page Frame Structures since these are linked in a double linked chain. **Warning:** Both the Dump Formatter and the Kernel Debugger may fail to recognize the chain pointers correctly. In particular the 2 high order-digits of the frame number are truncated. Use this option advisedly! See the Free Page Frame Structures or Idle Page Frame Structures for information on locating idle and free PF chains.

laddr Specifies the linear address of a specific Page Frame Structure to be formatted.

An address expression may be specified.

Ln Specifies the number of Page Frame Structures to display from the starting criterion.

frame Specifies a physical storage page frame number. This will cause the Page Frame Structure corresponding to that frame to be displayed except for UVIRT storage. PFs corresponding to UVIRT storage are zeroed unless aliased by non-UVIRT storage. In the former case .MP will display then next non-UVIRT PF. In the latter it will display the aliasing non-UVIRT PF. See the DP command for related information.

Results and Notes:

Page Frame Structures are allocated in contiguous storage from the address given by global variable:

_pft

Output from the **.**MP command appears in one of three formats.

In-use Page Frame Structure.
Idle Page Frame Structure.
Free Page Frame Structure.

For a description of the fields formatted by **.**MP see the **.**MP Output Field Descriptions.

For more examples using of the **.**M family of commands see Volume I, of the OS/2 Debugging Library 'Exploring Memory Management' chapter.

3.4.17.1 Free Page Frame Structures

```
ffdf509c Free:              BLink=0000f Flg=4 FLink=001da Blk=00001 Frame=0000d
ffdf50a8 Free:              BLink=001f2 Flg=4 FLink=0003f Blk=00001 Frame=0000e
ffdf50b4 Free:              BLink=001f1 Flg=4 FLink=0000d Blk=00001 Frame=0000f
ffdf50c0 Free:              BLink=001fe Flg=4 FLink=001f1 Blk=00000 Frame=00010
```

Figure 67. Free Page Frame Structures

Notes:

- Free Page Frame Structures are chained in a double-linked list. The head of this list may be located as follows:
  1. Locate list structure whose address is given by **_pgFreeList**
  2. The first double-word of the list structure points to the psuedo-page frame structure that heads the free list.
  3. The second double-word contains the pseudo-frame number of the pseudo-PF. N.B. This marks the end of the linked list only.
  4. The backward pointer to the first true free PF is given by the 5 low-order digits of the second double-word of the pseudo-PF. This value may be used with the **.**MP command.
- The **Blk** field has a residual field an is of no direct relevance to free Page Frame Structures.

For a description of the fields formatted by **.**MP see the **.**MP Output Field Descriptions.

3.4.17.2 Idle Page Frame Structures

```
ffdfcdb8 Idle:  pVP=ff1e6b9c Blink=01279 Flg=0 Flink=0004c Blk=0004a Frame=0127a
ffdfcdac Idle:  pVP=ff1e6ba8 Blink=01261 Flg=0 Flink=0127a Blk=0004b Frame=01279
ffdfcc8c Idle:  pVP=ff1e6c08 Blink=0125d Flg=0 Flink=01279 Blk=00066 Frame=01261
```

Figure 68. Idle Page Frame Structures

Notes:

- Idle Page Frame Structures are chained in a double-linked list. The head of this list may be located as follows:
  1. Locate list structure whose address is given by **_pgIdleList**
  2. The first doubleword of the list structure points to the psuedo-page frame structure that heads the idle list.
  3. The second double-word contains the pseudo-frame number of the pseudo-PF. N.B. This marks the end of the linked list only.
  4. The backward pointer to the first true idle PF is given by the 5 low order digits of the second double-word of the pseudo-PF. This value may be used with the **.**MP command.

For a description of the fields formatted by **.**MP see the **.**MP Output Field Descriptions.

3.4.17.3 In-use Page Frame Structures

```
ffdf5000 InUse: pVP=ff1df000 RefCnt=0001 Flg=0 ll=00 sl=00 Blk=00000 Frame=00000
ffdf500c InUse: pVP=ff1df060 RefCnt=0001 Flg=0 ll=00 sl=00 Blk=00000 Frame=00001
ffdf5018 InUse: pVP=ff1df06c RefCnt=0001 Flg=0 ll=00 sl=00 Blk=00000 Frame=00002
ffdf5024 InUse: pVP=ff1df078 RefCnt=0001 Flg=0 ll=00 sl=00 Blk=00000 Frame=00003
```

Figure 69. In-Use Page Frame Structures

For a description of the fields formatted by **.MP** see the .MP Output Field Descriptions.

3.4.17.4 .MP Output Field Descriptions

Output from .MP appears in one of is of the following forms:

```
ffdf500c InUse: pVP=ff1df060 RefCnt=0001 Flg=0 ll=00 sl=00 Blk=00000 Frame=00001

ffdfcdac Idle:  pVP=ff1e6ba8 Blink=01261 Flg=0 Flink=0127a Blk=0004b Frame=01279

ffdf50b4 Free:               BLink=001f1 Flg=4 FLink=0000d Blk=00001 Frame=0000f
```

Each of the fields has the following meaning:

address The linear address of the PF structure is given to the left of each display line.

type The type of PF is displayed in the second column. Three types are possible: Free, Idle and InUse.

pVP= The linear address of the associated Virtual Page Structure. See the **.**MV command for information on displaying Virtual Page Structures.

RefCnt The number of PTEs that reference the frame of physical storage represented by this PF. A reference count greater than 1 indicates shared memory, some instances of which will be represented by VMCOs (see the .MC command).

When a PTE is attached to an existing PF then the **Refcnt** is incremented.

When a page of memory is freed, the **Refcnt** is decremented. If it becomes zero the PF may be eligible for putting on the Idle list.

PFs corresponding to UVIRT storage are zeroed unless aliased by non-UVIRT storage. In either case no reference accounting is performed for UVIRT mappings.

Blink= The backward or right link to the previous Idle or Free PF.

Flink= The forward or left link to the next Idle or Free PF.

Flg= PF flags.

The following flags are defined:

Name	Bit mask	Description
PF_FAST	0x1	frame is fast memory
PF_BUSY	0x2	frame is busy
PF_FREE	0x4	frame is free
PF_RES	0x8	reserved

Notes: **PF_FAST** flag is set for some physical storage frames below 640K.

PF_BUSY signifies that access to the PF is being serialized by the page frame manager. This is normally followed by setting the **VP_BUSY** flag in the associated VP, if reset or setting the **VP_WANTED** flag and waiting on the the BlockID of the VP

address. Under the debug kernel the thread slot of the VP semaphore owner is saved in **vp_semowner** (VP+0x0a) See **.**PB command for information on thread slots waiting on BlockIDs.

ll= Count of number of long-term lock requests active against this page frame. This is incremented when a request to lock a range of pages of a memory object is made. It is also, but rarely, set to 1 to isolate page frames that have caused trap 2 errors from which the system has recovered. See also **.**MO command output for flags relating to memory object locking.

sl= Count of number of short-term lock requests active against this page frame. This is incremented when a request to lock a range of pages of a memory object is made. For related information, see **.**MO command output for flags relating to memory object locking.

Blk= Specifies the swap disk frame, loader block number or diagnostic flag depending on the flag settings of the corresponding Virtual Page Structure pointed to by the **pVP=** field.

When **VP_DF** is set and **VP_DISCARDABLE** is reset then **Blk=** is the swap disk frame number that contains a copy of the page frame.

When **VP_DISCARDABLE** is set and **VP_RESIDENT** is reset then the **Blk=** field is the Loader BlockID. Except for a special case noted below, this is a page index, starting from 1, into the objects of the module as an aggregated whole, with the size of each object rounded up to a page boundary. The special case occurs when the memory object that owes this page frame has an **hmte** set to system object id 0xffc0, **Discard Owner**. When this occurs the following special block numbers may be used:

0x0fffe System Infoseg
0x0fffd Local Infoseg
0x0fffc invalid LDT pages

When **VP_DF** and **VP_DISCARDABLE** are reset the **Blk=** usually indicates the last cross-linked swapper disk frame (unless its zero), however under the debug kernel negative values are used to indicate errors or instances where swapper frames have been freed because the corresponding PTE for the frame was found to be dirty. The following error indicators are possible:

-1 When also **Flg=9** then the physical frame caused a Trap 2 error, but the system was able to recover the data. The frame is isolated from further use by setting **ll=**1, **refcnt=**1 and **PF_FREE** flags are reset and **pVP=pgVPBasePg**
-3 A page-in operation failed with ERROR_SWAP_IO_PROBLEMS
-4 A page_out operation failed with PGPO_FAILED
-5 A page_out operation failed with ERROR_SWAP_FILE_FULL

Otherwise disk frame reclamation is indicated by **Blk=** values of: -1, -2, -7, -9 and 0xfff0.

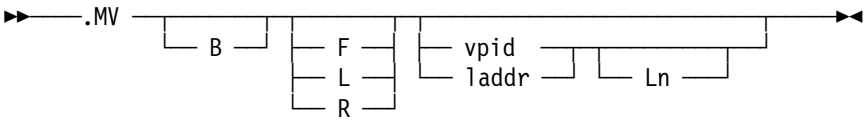
For related VP information, see the **.**MV command.

Frame= This is the physical page frame number that this Page Frame Structure represents.

3.4.18 .MV - Format Memory Virtual Page Structures (VPs)

Display memory Virtual Page Structures (VPs).

Syntax:



Parameters:

B Display in-use (busy) Virtual Page Structures in sequential order.

Note: In-use VPs are signified by a zero reference count and not by the **VP_BUSY** flag. See **Ref=** field in **.**MV Output Field Description.

F Display free Page Frame Structures.

L Follow left (forward) chain pointer. This is only of relevance to free Virtual Page Structures since these are linked in a double linked chain.

> **Attention**
>
> Both the Dump Formatter and the Kernel Debugger may fail to recognize the chain pointers correctly and under some circumstances the debug kernel may hang. Use this option advisedly!

See Free Virtual Page Structures for information on locating Free VP chains.

R Follow right (backward) chain pointer. This is only of relevance to Free Virtual Page Structures since these are linked in a double linked chain.

> **Attention**
>
> Both the Dump Formatter and the Kernel Debugger may fail to recognize the chain pointers correctly and under some circumstances the debug kernel may hang. (.MVFL will cause this effect). Use this option advisedly!

See Free Virtual Page Structures for information on locating Free VP chains.

laddr Specifies the linear address of a specific Virtual Page Structure to be formatted.

An address expression may be specified.

Ln Specifies the number of Virtual Page Structures to display from the starting criterion.

vpid Specifies a VP Id. This is an index in to the table of Virtual Page Structures, which are located in contiguous virtual storage.

Results and Notes:

Virtual Page Structures are allocated in contiguous storage from the address given by global variable:

_pgpVPBase

Output from the **.MV** command appears in one of two formats.

In-use Virtual Page Structure
Free Virtual Page Structure

For a description of the fields formatted by **.MV** see the **.MV** Output Field Descriptions.

For more examples using of the **.M** family of commands see Volume I, of the OS/2 Debugging Library 'Exploring Memory Management' chapter.

3.4.18.1 Free Virtual Page Structures

```
VPI=0d3e pVP=ff1e8ee8 free FLink=00000000 BLink=fff13280
VPI=0d3f pVP=ff1e8ef4 free FLink=ff1e9fec BLink=ff1e8cf0
VPI=0d40 pVP=ff1e8f00 free FLink=ff1e9fec BLink=ff1e8cf0
VPI=0d41 pVP=ff1e8f0c free FLink=00001000 BLink=02450030
VPI=0d42 pVP=ff1e8f18 free FLink=00000000 BLink=ff1e8f00
```

Figure 70. Free Virtual Page Structures

Notes:

- Free Page Frame Structures are grouped in bundles that are chained in a circular double link list. Each bundle comprises contiguous free VPs in the VP array. The chain pointers are only used by the head and tail of each bundle as follows:
  - For bundles of greater than one VP:
    1. **Blink** of the head points to the tail
    2. **Flink** of the head points to the head of the next bundle
    3. **Blink** of the tail points to the head of the previous bundle
    4. **Flink** of the tail is set to zero
  - For single VP bundles:
    1. **Blink** points to the head of the previous bundle
    2. **Flink** points to the head of the next bundle

  The free VP chain is headed by a pseudo-VP whose **Blink** points to the head of the first true free bundle and whose **Flink** points to the last VP in the VP array. The pseudo-VP is located at global symbol:

  _pgVPHead

- Unless a free VP is the head or tail of a bundle the **Flink** and **Blink** will retain values from its previous use. In particular it may be possible to glean information about a previous allocation at the **Flink** field overlays the **Flg** and **Block** fields and the **Blink** field overlays the **HobPg** and **Hob** fields of an in-use VP. In the example above VPI d41

was probably allocated to page 30 of hob 245. Using the following **.MO** command might reveal who the owner was and who allocated this storage.

```
.MOC 245
```

For a description of the fields formatted by **.MV** see the **.MV** Output Field Descriptions.

For more examples using of the **.M** family of commands see Volume I, of the OS/2 Debugging Library 'Exploring Memory Management' chapter.

3.4.18.2 In-use Virtual Page Structures

```
VPI=0000 pVP=ff1df000 Res Frame=0000 Flg=410 HobPg=0000 Hob=ff77 Ref=001 Own=000
VPI=0001 pVP=ff1df00c Res Block=0000 Flg=c00 HobPg=0000 Hob=ff6c Ref=042 Own=000
VPI=0002 pVP=ff1df018 Res Frame=0bc5 Flg=410 HobPg=0000 Hob=0001 Ref=001 Own=000
VPI=0003 pVP=ff1df024 Res Frame=0bc4 Flg=410 HobPg=027a Hob=0022 Ref=001 Own=000
```

Figure 71. In-Use Virtual Page Structures

For a description of the fields formatted by **.MV** see the **.MV** Output Field Descriptions.

3.4.18.3 .MV Output Field Descriptions

Output from **.MV** appears in one of is of the following forms:

```
VPI=0000 pVP=ff1df000 Res Frame=0000 Flg=410 HobPg=0000 Hob=ff77 Ref=001 Own=000
VPI=0001 pVP=ff1df00c Res Block=0000 Flg=c00 HobPg=0000 Hob=ff6c Ref=042 Own=000

VPI=0d40 pVP=ff1e8f00 free FLink=ff1e9fec BLink=ff1e8cf0
```

Each of the fields has the following meaning:

VPI=
The VP index into the array of VPs.

pVP=
The linear address of the VP.

status
The status of the VP interpreted from the **Flg** field. The following values may appear:

SOW	Swap on Write flag (VP_SOW set)
Res	Page is resident (VP_RESIDENT set)
Dsc	Page is discardable (VP_DISCARDABLE set)
Swp	Page is swappable (VP_DISCARDABLE reset)
free	VP is free (vp_refcount=0)

Block=*nnnn*
The cross-linked loader block number or swapper disk frame. This implies the virtual page is not attached to a PF. If it is:

discardable Then it is linked to a loader BlockID,

swappable Then it is linked to a swapper disk frame.

When the page is swappable (VP_DISCARDABLE reset) and does not have a disk frame (VP_DF reset) then the following special **Block** values may be used:

0 Allocate PF on demand
1 Allocate on demand zero-fill page
2 page is in a broken disk frame

Frame=*nnn*
The virtual page is linked to PF ***nnnn***. Refer to the **.**MP command for displaying information about the related page frame.

Flink=
Forward link of a free VP. This is only of relevance to the VP at the head of a bundle of free VPs. See Free Virtual Page Structures for information on how free VPs are linked.

Blink=
Backward link of a free VP. This is only of relevance to the VP at the head and tail of a bundle of free VPs. See Free Virtual Page Structures for information on how free VPs are linked.

Flg=
VP flags.

The following flags are defined:

Name	Bit mask	Description
VP_BUSY	0x001	page semaphore taken
VP_WANTED	0x002	page semaphore requested
VP_CACHE	0x004	search page cache for pf
VP_PFIDLE	0x008	cross linked to idle pf
VP_PF	0x010	cross linked to pf
VP_DF	0x020	has swap file disk frame
VP_DIRTY	0x040	contents written to - from pte
VP_SHDIRTY	0x080	shadow dirty bit (for VDMs)
VP_SOW	0x100	change to swappable on write
VP_PRIVATIZED	0x200	vp privatized
VP_RESIDENT	0x400	cannot be moved - value from pte
VP_DISCARDABLE	0x800	1 = discardable, 0 = swappable

Notes: **PF_BUSY** signifies that access to the VP is being serialized by the page frame manager.

VP_WANTED signifies that a thread is waiting to mark the VP busy. The thread will wait on a BlockID of the VP address. Under the debug kernel the thread slot of the VP semaphore owner is saved in **vp_semowner** (VP+0x0a) see **Own=** field of the **.**MW command. See **.**PB command for information on thread slots waiting on BlockIDs.

HobPg=
The relative page number of the memory object that this VP is assigned to. See **Hob=** field below.

Hob=

The hob of the memory object to which this page is assigned.

Note: Use

```
.MOC hob
```

to obtain the virtual address and owner information relating to this VP. See the **.**MO command for more information.

Ref=

The number of memory objects sharing this page of data. A reference count greater than 1 indicates shared memory, some instances of which will be represented by VMCOs, (see the **.**MC command) and others by aliases (see the **.**ML command).

The reference count is incremented and decremented according to usage. When the count becomes zero the VP is no longer in use and any committed physical storage or swapper storage my become eligible for freeing.

UVIRT storage is not represented by VPs thus reference accounting is not performed.

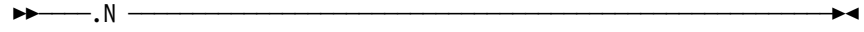
Own=

The thread slot number of the current owner of the VP semaphore. This field is only used in the debug kernel and will only have significance if the **VP_BUSY** or **WP_WANTED** flags are set. See **.**PB command for information on thread slots waiting on BlockIDs.

3.4.19 .N - Display Dump Information Summary

Display information saved by from the operating system when the stand-alone dump procedure was initiated.

Syntax:



Parameters:

None

Results and Notes:

.N command displays information saved when the kernel routine **RASRST** is entered at sand-alone dump initiation.

.N displays the following information:

```
gdtr_lim: 1FFF
gdtr_base: 7C3E5000
idtr_lim: 03FF
idtr_base: FFE00150
ldtr_reg: 0028
lo_data_sel: 0400
hi_data_sel: 0400
trace_buf_addr: 0B490400
sys_anchor_sel: 0070
arena_base: FEB1F020
max_threads: 0101
phys_page_dir: 001D6000
vm_object_ptr: FEC80020
StartInit_Data: 00000140
dcm_ote_start: FFF0A92F
CurProcPid: 000D
TaskData: 0B5C0400
FirstPacket: 158A
LastPacket: 04C0
SysSemDataTable: 53A60400
GDT_Buffers: 00A80138
PapTCBPtrs: 0B6B0400
callerSS: 00E8
callerESP: 00000FCC
savePage: 00241467
```

Each of the items displayed has the following significance:

gdtr_lim:
The current GDTR register limit value.

gdtr_base
The current GDTR register base address.

ldtr_lim
The current LDTR selector limit.

ldtr_base
The current LDTR selector base address.

ldtr_reg
The current LDT selector.

lo_data_sel
Selector for DOSGROUP segment.

hi_data_sel
Selector for DOSHIGHDATA segment.

trace_buf_addr
Offset:selector address of **ras_stda_addr**; the selector for the system trace buffer.

sys_anchor_sel
Selector for the SAS.

sarena_base
Value of **_parvmOne**, the pointer to the first VMAR.

max_threads
Maximum Thread Slot Number.

phys_page_dir
Value of cr3 register (that is, the physical address of the page directory table).

vm_object_ptr
Value of **_pobvmOne**, the pointer to the first VMOB

StartInit_data
Value of **_StartINITData**.

dcm_ote_start
Address of DOSCALLS.DLL OTE.

CurrProcPid
Current process ID.

TaskData
Offset:selector address of scheduler global data.

FirstPacket
First word of the first device driver strategy 2 request packet in packet pool.

LastPacket
First word of the last device driver strategy 2 request packet in packet pool.

SysSemDataTable
Offset:selector address of **SysSemDataTable**.

GDT_Buffers
GDT selector for buffer segment. The low-order word of this field should be ignored.

PapTCBPtrs
Offset:selector address of **papTCBPtrs**. The word value at this label is an offset from the DOSGROUP selector (400) to the thread slot table.

callerSS

The **SS** selector on entry to **RASRST**, the stand-alone system dump entry point within the kernel.

Note: If this dump was taken in interrupt mode the **SS** selector will be **E8**. Further more, if that last device driver to use the interrupt stack is **KDB$** then it is possible the the dump process was initiated with the **Ctrl-Alt-Numlock-Numlock** sequence. If the dump was taken in kernel mode then the **SS** selector will probably be **30** and the dump will have been initiated because of a trap or call to **DosForceSystemDump** API.

callerESS

The **ES** register on entry tp **RASRST**, the stand-alone system dump entry point within the kernel.

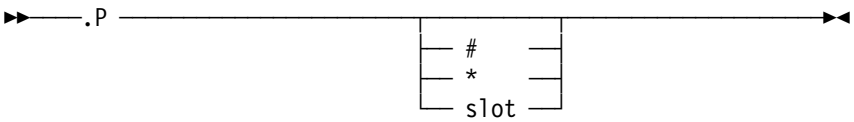
savePage

Page directory entry 0. This data is overwritten by the dump process.

3.4.20 .P - Display Process Status

Display process and thread status information from the Per Task Data Area (PTDA), Thread Control Block (TCB) and Thread Swappable Data (TSD).

Syntax:



Parameters:

slot

Display process status for thread slot ***slot***.

The following shorthand may be used for the slot number:

***** The current (last) thread the dispatcher gave control to. This value is taken from the word a global label:

_TaskNumber

The debugger default thread slot. This defaults to the current slot unless overridden by the **.**S command.

If no slot number is given then all thread slots are displayed and grouped by process.

Results and Notes:

The **.**P command locates a thread's TCB from either the thread slot table, the linear address of which is given by the following global variable:

_papTCBSlots

or by traversing the process tree using **TCBpTCBNext** (**TCB** +0x14), **TCBpPTDA** (**TCB** +0x08) and **ptda_pTCBHead** (**PTDA** + 0x20) fields. Output from the **.**P command appears in tabular form as follows:

```
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
 0001  0001 0000 0000 0001 blk 0100 ffe3a000 ffe3c7d4 ffe3c61c 1eb4 00 *ager
 0002  0001 0000 0000 0002 blk 0200 7b7ca000 ffe3c7d4 7b9c8020 1f3c 00 *tsd
 0003  0001 0000 0000 0003 blk 0200 7b7cc000 ffe3c7d4 7b9c81d8 1f50 00 *ctxh
 0004  0001 0000 0000 0004 blk 081f 7b7ce000 ffe3c7d4 7b9c8390 1f48 00 *kdb
 0005  0001 0000 0000 0005 blk 0800 7b7d0000 ffe3c7d4 7b9c8548 1f20 00 *lazyw
 0006  0001 0000 0000 0006 blk 0800 7b7d2000 ffe3c7d4 7b9c8700 1f3c 00 *asyncr
*0008  0002 0001 0002 0001 blk 0500 7b7d6000 7b9e4020 7b9c8a70 1eb8 01 pmshell
 000a# 0002 0001 0002 0002 blk 0800 7b7da000 7b9e4020 7b9c8de0 1ed4 01 pmshell
```

Figure 72. Command .P Output

Each of the fields has the following meaning:

slot

The unique (hexadecimal) index in to the thread slot table of all threads.

This value may be flagged with:

***** To the left to signify the current (or last) dispatched thread.
# To the right to signify the Kernel Debugger or Dump Formatter default thread slot.

Slot may be found in the **TCBNumber** (**TCB** + 0x2) field of the **TCB**

Pid

The process id this thread slot is assigned to.

Ppid

The parent process id that created this thread. A value of zero signifies a detached process.

Csid

The command subtree id.

This is normally the same value as the **Pid**. When the parent process dies any orphaned children are adopted by their grandparent by making **Ppid** equal to the grandparent's **Pid**. Each orphan inherits the **Csid** of its dying parent. This mechanism ensures that orphaned PTDAs are not retained for returning termination information to their parent (via DosWaitChild).

Csid is taken from the **Csid** (**PTDA** +0x4be (H/R: +0x4b6)) field of the **PTDA**.

Ord

The thread ordinal for this thread slot. This is the unique thread id assigned to the thread within the process to which it belongs.

Ord is taken from the **TCBOrdinal** (**TCB**+ 0x0) field of the **TCB**

Sta

The thread's ascending scheduler state taken from the **TCBState** (**TCB** +0x161) field.

Except when a state transition is progress this is the same as the current state of the thread (see the **Qst** field of the **.**PQ command.)

The following states are possible:

Table 9. Process States

Abbreviation	State	TCBState value	Description
---	STATE_VOID	0	Uninitialized or Dead thread
rdy	STATE_READY	1	Thread ready to run
blk	STATE_BLOCKED	2	Blocked on a BlockID
sus	STATE_SUSPENDED	3	*** Not in Use ***
crt	STATE_CRITSEC	4	Blocked by another CritSec thread (after attempting to run)
run	STATE_RUNNING	5	Thread currently running
bst	STATE_READYBOOST	6	Ready, but apply an IO boost after swapping in a TSD
tsd	STATE_TSD	7	Thread waiting for the TSD daemon to page in the TSD.
dly	STATE_DELAYED	8	Delayed TKWakeup (Almost Ready)
frz	STATE_FROZEN	9	Frozen Thread via DosSuspendThread, DosCreateThread, DosExecPgm or DosSystemService
gsk	STATE_GETSTACK	10	TSD daemon is waiting for the page manager to page-in a TSD
bad	STATE_BADSTACK	11	TSD failed to page-in

Notes: The scheduler manages threads on queues by priority and state. See the **.**PQ command for displaying scheduler queues.

The scheduler uses a finite state machine to manipulate thread queues. **TCBQState** and **TCBState** are the state transition drivers. They hold a thread′s current and desired state. Except during a state transition current and desired state will be identical.

STATE_RUNNING is set when the next potential runner has been selected. The running thread′s context is then switched and various dispatcher flags checked before finally giving control to user code. It is therefore possible for the running state to be set and for the user code not to run.

STATE_READYBOOST is a modified ready state and never becomes the current state, instead a priority boost is applied and the state becomes **STATE_READY**.

STATE_CRTSEC state applies to non-critical section threads only. It is only set when a critical section thread within the same process has given up the processor, while still in critical section, and another thread in the same process is selected to run. If this thread is thread 1 of the process and there are pending signals to process, the thread′s signal handler will be dispatched. When there are no more pending signals or this thread is not thread 1, then **STATE_CRITSEC** will be set.

STATE_FROZEN is normally only seen when an application uses the DosSuspendThread API or creates a thread (or process) that is initially suspended. DosSystemService is used by the session manage to freeze all threads of a process in one system call.

Many states are transient accordingly the persistent appearance of a particular state might indicate a problem of the following nature:

rdy Many ready threads might indicate contention for processor time. Tends to indicate the existence of a higher priority CPU-bound thread.

run Under the Dump Formatter this would indicate a trapped or processor-bound thread. Under the Kernel Debugger a processor bound thread.

blk All threads blocked could indicate no-work or a deadlock. Under Dump Formatter this would imply a manually invoked dump using Ctrl-Alt-Numlock-Numlock or use of the DosForceSystemDump API.

--- The void state is rarely seen. Under Dump Formatter this probably indicates an incorrect version of the Dump Formatter for the system dumped.

crt Another thread in the same process is in critical section and is either blocking without exiting critical section or is processor bound.

dly Another thread is processor bound.

frz A deadlock, loop or no-work for the freezing thread.

sus Is not used, so probably indicates a mismatch between the Dump Formatter and dump.

tsd Physical storage overcommitted. Swapper very large. System may be thrashing.

bst Physical storage overcommitted. Swapper very large. Very busy processor bound system System may be thrashing.

gsk Physical storage overcommitted. Swapper very large. Very busy processor bound system System may be thrashing.

bad Excessive swapper. System may die. Physical storage overcommitted. Very busy processor bound system.

Pri Thread priority (word length field) in **TCBPriority** (**TCB** +0x168).

This is the current priority calculated by the scheduler based upon priority class (**TCBPriClass** (**TCB** +0x164)), priority class level (**TCBPrilevel** (**TCB** +0x165)) and priority boosts (**TCBPriClassMod** (**TCB** +0x166)).

The following priority classes are defined:

Class	Value	Description
CLASS_IDLE_TIME	0x01	Idle-Time class
CLASS_REGULAR	0x02	Regular class

Class	Value	Description
CLASS_TIME_CRITICAL	0x03	Time-Critical class
CLASS_SERVER	0x04	Client/Server Server class

The following priority boosts (class modifiers) are defined:

Boost	Value	Description
CLASSMOD_KEYBOARD	0x04	Keyboard boost
CLASSMOD_STARVED	0x08	Starvation boost
CLASSMOD_DEVICE	0x10	Device I/O Done Boost
CLASSMOD_FOREGROUND	0x20	Foreground boost
CLASSMOD_WINDOW	0x40	Window Boost
CLASSMOD_VDM_INTERRUPT	0x80	VDM simulated interrupt boost

Note: **CLASSMOD_KEYBOARD** has no effect on **CLASS_SERVER**

The priority level is a value between 0x0 and 0x1f.

Priority class and modifier values are logically ORed to form an index into the priority class translation table, which is located at global symbol:

_schPriClassTbl

The resulting value is logically ORed with the priority level. The final value is subject to the minimum thread priority (**TCBPriorityMin** (**TCB** +016a)).

Priority boosts do not affect the priority of idle and time-critical threads.

Priority level has little or no effect on the priority of boosted regular and server class threads. threads

pTSD

Linear address of the TSD control block associated with this thread this thread taken from the **TCBpTSD** (**TCB** +0x0c).

Note:

The TSD contains the ring 0 stack for the associated thread. For the current thread this is addressable from selector 30 however the base address of selector 30 is entirely different from **TCBpTSD.** This is because the two addresses are aliased using two PTEs to pin the same physical frame. This device allows the TSD for be accessed out-of-context by the system, at the same time protecting system code from erroneous stack references.

pPTDA

Linear address of the PTDA control block representing the process to which this thread belongs. The address is taken from **TCBpPTDA** (**TCB** +0x08).

pTCB

Linear address of the TCB control block which represents the thread.

Note:

The output from **.P** is ordered by process and child process. TCBs are initially located from the thread table then the chain pointer **TCBpTCBNext** (**TCB** +0x14) is used to locate the remaining threads of a process.

Under the Dump Formatter **.P** will occasionally miss a thread because of the non-sequential manner in which the thread table slots are re-used. To ensure all active threads are displayed use **.PU**, **.PB** or **.PQ** commands.

Disp

The displacement into the TSD for the current thread that the dispatcher will use for its ESP after having switched back to this thread's context.

This value is calculated from **TSDKernelESP**; it therefore requires the **TSD** to be present. If the **TSD** is not present then a blank value is given. The **TSD** may be forced present under the Kernel Debugger by use of the **.I** command.

SG

Screen Group ID currently assigned to this process.

The Screen Groups ID is taken from the console locus structure (**Cons_Loc +0x2)** embedded in the PTDA ((**PTDA** +0x526 (H/R: +0x51e))).

Name

The name of the primary executable running in this process.

Except for process 1 and DOS Virtual Machines the name is obtained from the hmte stored in **ptda_module** (**PTDA** +0x5a6 (H/R: +0x59e)). If the SMTE is paged in then the name is taken from the file name pointed to by **smte_path** otherwise it it taken from mte_module and prefixed with an ! point. See the **.LM** command for information on formatting loader control blocks.

Process 1 comprises internal threads, that is threads which run in the kernel and are not separately loaded modules. **ptda_module** is zero for this process so the Dump Formatter and Kernel Debugger translate the Tids for Pid 1 as follows:

Tid	Name	Description
1	*ager	Ager thread used for compressing the Swap File.
2	*tsd	Scheduler's Daemon Thread used to page in TSDs
3	*ctxh	Default Global Context Hook dispatching thread.
4	*kdb	Kernel Debugger Daemon thread used to process page-in requests from the **.ID** command
5	*lazyw	File system cache lazy writer thread.
6	*asyncr	File system asynchronous read ahead thread.
7	*sysinit	System initialization thread.
8-n		Other transient internal threads associated with system initialization have a blank name.

Virtual DOS Machines run the DEM component of OS/2 to provide DOS emulation. DOS programs are loaded by the DEM and not known to the (OS/2) loader. Thus ptda_module is zero and the Kernel Debugger and Dump Formatter use the name ***vdm** to indicate a VDM. The PSP of the first loaded DOS program in a process may be located from **CurrentPDB** (**PTDA**

+0x2ea), which contains its segment address. The preceding paragraph contains the DOS arena record, the last 8 bytes of which contains the DOS program name currently executing.

3.4.20.1 Scheduler Finite State Machine

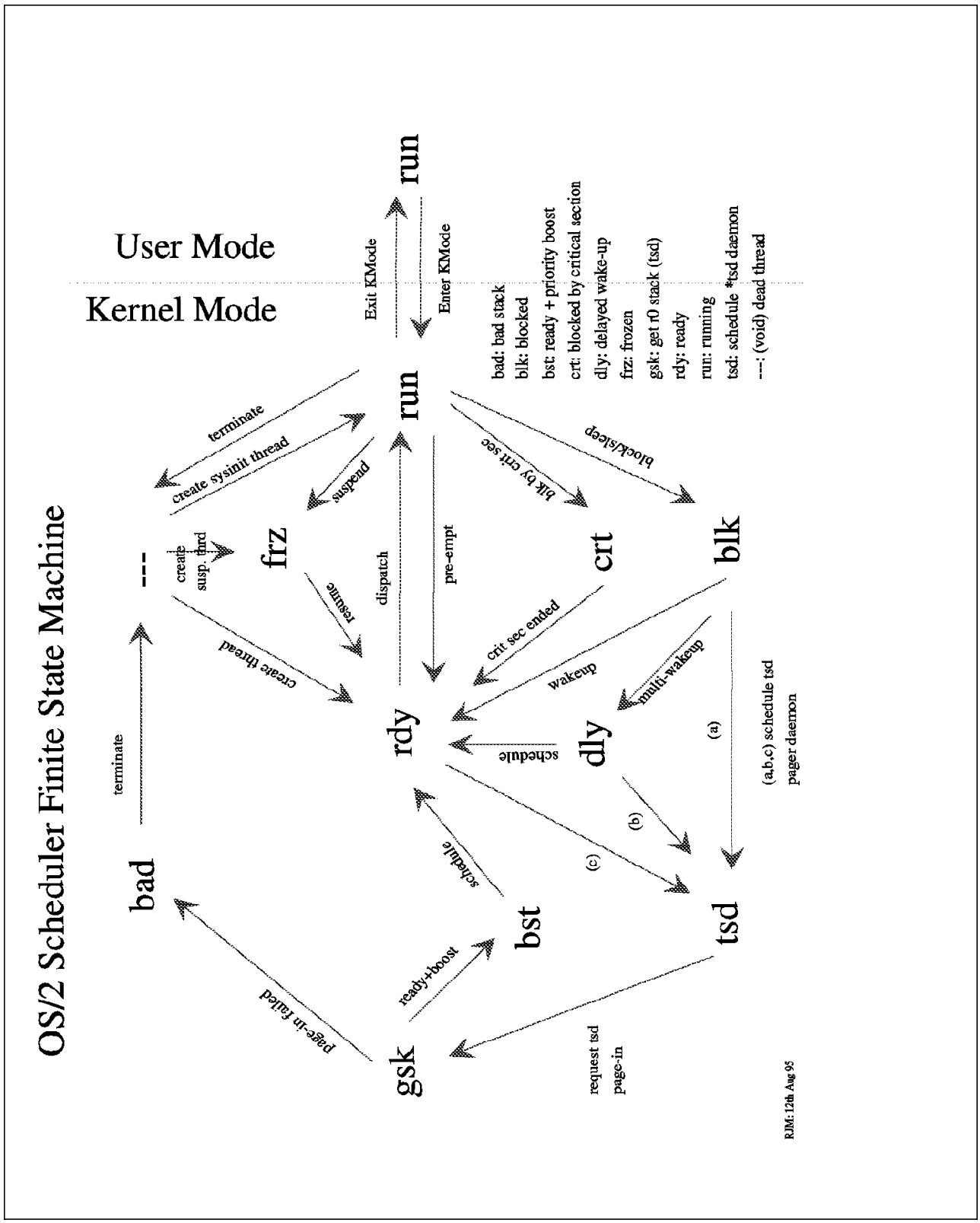
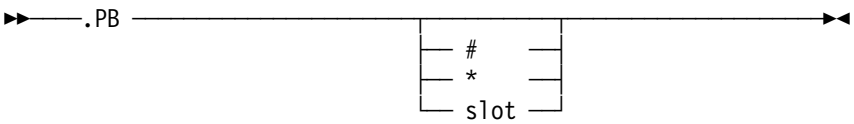


Figure 73. Scheduler Finite State Machine

3.4.21 .PB - Display Blocked Thread Information

Display information about all blocked threads.

Syntax:



Parameters:

slot

Display user information for thread slot ***slot***.

The following shorthand may be used for the slot number:

***** The current (last) thread the dispatcher gave control to. This value is taken from the word a global label:

_TaskNumber

The debugger default thread slot. This defaults to the current slot unless overridden by the **.**S command.

If no slot number is given then all thread slots are displayed in slot number order.

Results and Notes:

The **.**PB command locates each thread's TCB from the thread slot table, the linear address of which is given by global variable:

_papTCBSlots

or by traversing the process tree using **TCBpTCBNext** (**TCB** +0x14), **TCBpPTDA** (**TCB** +0x08) and **ptda_pTCBHead** (**PTDA** + 0x20) fields.

Output is displayed only if a thread is blocked on a BlockID. It appears in tabular form as follows:

```
Slot  Sta BlockID  Name     Type        Addr    Symbol
0001  blk fff11050 *ager
0002  blk fff74f59 *tsd                        _tkTSDDaemon
0003  blk fff43c78 *ctxh                       _kmCTXHDaemon
0004  blk fff7545a *kdb                        _tkKDBDaemon
0005  blk fff02dfc *lazyw                      _semLW
0006  blk fff111d4 *asyncr                     _AsyncReadSem
0008  blk fffe000e pmshell  RamSem    074b:06d6
000a  blk ffca0002 pmshell
000b  blk fffd000b pmshell  MuxWait
000c  blk fffd000c pmshell  MuxWait
000d  blk 04000df0 pmshell  DosSem    0400:0df0 CtrlNumLkQ
0007  blk fe750a10 pmshell  Sem32     8001 0019 vhevLazyWrite
0010  blk fe728dcc pmshell  Sem32     8001 0001 SrvReq
0011  blk fffe0006 pmshell  RamSem    d0c7:0020
```

```
 0012  blk fffd0012 pmshell  MuxWait
*0013  blk fffe0007 pmshell  RamSem    d09f:0bc0 memory_pool + 127
 0014# blk fffe0008 pmshell  RamSem    d09f:0bc8 memory_pool + 12f
```

Each of the fields has the following meaning:

slot
The unique (hexadecimal) index in to the thread slot table of all threads.

This value may be flagged with:

***** to the left to signify the last dispatched thread.
# to the right to signify the Kernel Debugger or Dump Formatter default thread slot.

Slot may be found in the **TCBNumber** (**TCB** + 0x2) field of the **TCB**

Sta
The ascending or desired state of the thread. This should always appear as **blk** for the **.PB** command, however Dump Formatter does not check the thread state so formats all threads. Those whose state is not **blk** should be ignored. See 3.4.20.1, “Scheduler Finite State Machine” on page 245 and the .PQ command for more information on thread states.

BlockID
The token used by **TKSleep** and **TKWapeUp** to sleep and wake a thread on an event.

The **BlockId** is taken from **TCBSleepID** (**TCB** + 0x18c).

The **BlockID** is a conventional value. A number of conventions are used by various system components. Usually the **BlockID** is constructed so to be unique across all conventions. Frequently it will refer to the address of an associated resource, such as a system control block, or a field within a control block. See the discussion of the **Type** field below for more information on interpreting **BlockIDs**.

Name
The name of the primary executable running in this process.

See **name** field description of the .P command for further information.

Type
Interpretation of the use of the **BlockID** in conjunction with double word **TCB_SemInfo** (**TCB** + 0x14c) and double word **TCB_SemDebugAddr** (**TCB** + 0x150).

The following **Types** are recognized by the Dump Formatter and Kernel Debugger:

RamSem
The thread is waiting on a **RamSem** or **FastSafeRamSem**.

The high word of the **BlockID** is **0xfffe**; the low word is the **RamSemID** taken from the RamSem structure.

The **Addr** field is taken from **TCB_SemInfo**. This is a ***selector:offset*** address of the **RAMSEM**. The **RamSem** may be imbedded within a FastSafeRamSem or a PMFastSafeRamSem.

The **Symbol** displayed is either that of the **TCB_SemDebugAddr** or if -1, the **RamSem** address. See the LN command for information on displaying symbols.

MuxWait

The thread is waiting on multiple events.

The high word of the **BlockID** is **0xfffd**, the low word is the **slot** of the waiting thread.

TCB_SemInfo and **TCB_SemDebugAddr** are not used with a **MuxWait BlockID**.

To locate the semaphores that comprise a given **MuxWait** proceed as follows:

1. Locate the **MuxWait** table at symbol **MuxTable**. Display this using **DB** for convenience. This table comprises 9 byte entries whose format is given by the MuxTableEntry structure.
2. Scan the **MuxTable** for entries that have this thread's slot number (+0x2 into each entry).
3. Of those entries, select those with non-zero **MuxType** (+4 into each entry).
4. Choose one of the following:

   For type 1 (SysSem)
   The last double word is the linear address of a system semaphore structure. Use the technique described below under **SysSem** for interpreting the **SysSem**.

   For type 2 (RamSem)
   The last double word of the entry contains the **RamSem** handle, the high word is the hob of the memory object containing the **RamSem**. The low word is the offset into the object where the **RamSem** is located. Use the technique described above under **RamSem** for interpreting the **RamSem**

   For type 3 (Physical RamSem)
   The last double word is the physical address of the **RamSem.**

   For type 4 (32-bit event sem)
   The last double word is the physical address of a 32-bit event semaphore. See **Sem32** below.

See Volume I of *The OS/2 Debugging Library*, example debugging log for an explicit example of using this technique.

Addr and **Symbol** fields are blank.

ReqPkt

The thread is waiting for an I/O request packed to complete.

The **BlockID** is the ***Selector:Offset*** address of the request packet. The Selector is the DOSGROUP kernel selector and should be selector 400.

The address should lie between addresses at global symbols: **FirstPacket** and **LastPacket**.

See the Physical Device Driver Reference manual for information on device driver request packets.

Addr and **Symbol** fields are blank.

SysSem

The thread is waiting on a system semaphore.

The **BlockID** is the ***Selector:Offset*** address of the SysSemTblStruc structure. The Selector is the DOSGROUP kernel selector and should be selector 400.

The address should lie within the System Semaphore Data Table, located at symbol **SysSemDataTable** for length 256*6 bytes.

Offset +0 of each table entry contains the owner's thread number.

The name associated with the semaphore may be located as follows:

1. Locate the **SysSem** RMP segment by displaying doubleword at symbol **SysSemRMPHdl**. The high word is the selector for the semaphore RMP.
2. Display the System Semaphore RMP using **DB**. The first 0x14 bytes is the RMP header. The remainder comprises variable length records. The first word of each record is its length and therefore the relative offset to the beginning of the following record. Offset 2 of each record is the semaphore data table selector offset.
3. Scan the RMP looking for an offset that matches the low word of the **BlockID**. When found the remaining bytes of the RMP record is the semaphore name (with the top two bytes overlaid by the semaphore offset).

See Volume I of *The OS/2 Debugging Library*, example debugging log for an explicit example of using this technique.

The **Addr** and **Symbol** fields are blank.

DosSem

The thread is waiting on an internal **RamSem**.

The **BlockID** is the ***selector:offset*** of the **DosSem**. The Selector is the DOSGROUP kernel selector and should be selector 400. The offset does not lie in the System Semaphore Data Table of the I/O Request Packet Table.

Addr is the **BlockID** formatted as ***selector:offset***.

The **Symbol** displayed is either that of the **TCB_SemDebugAddr** or if -1, the **DosSem** address. See the LN command for information on displaying symbols.

Sem32

The thread is waiting on a 32-bit semaphore.

The **BlockID** is the address of the 32-bit Semaphore structure.

TCB_SemInfo contains the semaphore handle. This is of the form:

8001nnnn Shared 32-bit semaphore. ***nnnn*** is the (doubleword) index into the shared semaphore table located at symbol **_pShSemTbl**. Each entry is an address of the corresponding 32-bit semaphore structure. (That is, the **Sem32 BlockID**).

0001nnnn Private 32-bit semaphore. ***nnnn*** is the (doubleword) index into the private semaphore table located at **pPrSemTbl** (**PTDA** +0x4cc (H/R: +0x4c4)). Each entry is an address of the

corresponding 32-bit semaphore structure. (That is, the **Sem32 BlockID**).

Addr field is the semaphore handle formatted as two words.

The **Symbol** displayed is either that of the **TCB_SemDebugAddr** or if -1, the **Sem32** address. See the LN command for information on displaying symbols.

Use the **.**D SEM32 command with the **BlockID** to format the 32-bit semaphore.

Buffer

The thread is waiting for a file system buffer.

The **BlockId** is the ***selector:offset*** address of the buffer. The high word is the buffer selector and should be **a8**.

The **Addr** and **Symbol** fields are blank.

SFT

The thread is waiting for a SFT entry.

The **BlockId** is the ***selector:offset*** address of the SFT. The high word is the buffer selector and should be one that is listed in the **SFT** table pointed to by **c0:0**.

The **Addr** and **Symbol** fields are blank.

ChildWait

The thread is waiting in DosWaitChild for a child process to terminate.

The high word of the **BlockID** is the **ptda_Pid** offset from selector 30 (0xffca).

The low word of the **BlockID** is the Pid to which this thread belongs.

blank type

The thread is waiting on a **BlockId** that the Kernel Debugger and Dump Formatter have not been able to identify.

Addr field is blank.

The **Symbol** displayed is either that of the **TCB_SemDebugAddr** or if -1, the **Sem32** address. See the LN command for information on displaying symbols.

Notes:

The **BlockID** interpretation is not exact. A device driver, for example, could call **DevHlp_ProcBlock** using a value for **BlockID** that conflicts with another convention.

Under the Debug kernel only, **TCB_SemDebugAddr** is used to record the creator's address of kernel, system and RAM semaphores. If it is not used it is set to 0xffffffff.

ChildWait semaphores might be missed by the Dump Formatter and Kernel Debugger. Look out for **BlockIDs** of the form **0xffca????**.

Some **Sem32 BlockIDs** are missed by the Dump Formatter. Check **TCB_SemInfo** for a 32-bit semaphore handle and **BlockIDs** of the form **0xfe??????**

If **BlockID** is a linear address owned by ksem then the semaphore is a **Kernel Semaphore**. However, not every **KSEM** is owned **ksem** owned memory. Under the ALLSTRICT kernel, a **KSEM** may be readily identified from the first 4 bytes, which have the signature ″**KSEM**″ Use **.**D KSEM command against the **BlockID**.

In general a **BlockID** will be chosen to be meaningful to the programs using it. Often it is an address of a resource that needs to be serialized. Where no other information is given one should try:

.M BlockID	To try to establish an owner of the resource represented by the **BlockID**
LN BlockID	To try to establish a meaningful symbol associated with the **BlockID**
Unwind User′s Stack	To try to establish the API or call the lead to the thread waiting on the **BlockID** (see the **.**K command)

Addr
The address of the semaphore associated with this **BlockID**

See **Type** field discussion above for more precise information.

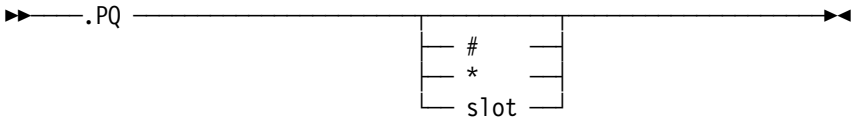
Symbol
Either the symbolic address of the creator or of the associated semaphore.

See **Type** field discussion above for more precise information.

3.4.22 .PQ - Display Scheduler Queue Information

Display scheduler thread queue information for all (active) threads.

Syntax:



Parameters:

slot
Display queue status for thread slot ***slot***.

The following shorthand may be used for the slot number:

***** The current (last) thread the dispatcher gave control to. This value is taken from the word a global label:

_TaskNumber

The debugger default thread slot. This defaults to the current slot unless overridden by the .S command.

If no slot number is given then all thread slots are displayed in slot number order.

Results and Notes:

The **.**PQ command locates each thread's TCB from the thread slot table, the linear address of which is given by global variable:

_papTCBSlots

or by traversing the process tree using **TCBpTCBNext** (**TCB** +0x14), **TCBpPTDA** (**TCB** +0x08) and **ptda_pTCBHead** (**PTDA** + 0x20) fields.

Output from the **.PQ** command appears in tabular form as follows:

```
 Slot  QSt Pri  pTCB     PriNextQ PriPrevQ PriHigh  PriLow   PriNext  PriPrev
 0001  blk 0100 ffe3c61c
 0002  blk 0200 7b9c8020
 0003  blk 0200 7b9c81d8
 0004  blk 081f 7b9c8390
 0005  blk 0800 7b9c8548
 0006  blk 0800 7b9c8700 7b9cb3b0 7b9c9830
 0008  blk 0500 7b9c8a70
 000a  blk 0800 7b9c8de0
*000b  blk 0800 7b9c8f98 7b9ca960 7b9ca960
 000c# blk 0800 7b9c9150 7b9cab18 7b9cab18
```

Each of the fields has the following meaning:

slot
The unique (hexadecimal) index in to the thread slot table of all threads.

This value may be flagged with:

***** To the left to signify the current (or last) dispatched thread.
# To the right to signify the Kernel Debugger default thread slot.

Slot may be found in the **TCBNumber** (**TCB** + 0x2) field of the **TCB**.

QSt
The thread's descending or current scheduler state taken from the **TCBQState** (**TCB** +0x160) field.

Except when a state transition is progress this is the same as the desired state of the thread (see the **Sta** field of the **.**P command.)

The following states are possible:

Table 10 (Page 1 of 2). Thread States and Description

Abbreviation	Qstate	TCBQState Value	Description
---	STATE_VOID	0	Uninitialized or Dead thread
rdy	STATE_READY	1	Thread ready to run
blk	STATE_BLOCKED	2	Blocked on a BlockID
sus	STATE_SUSPENDED	3	Suspended by DosSuspendThread

Table 10 (Page 2 of 2). Thread States and Description			
Abbreviation	**Qstate**	**TCBQState Value**	**Description**
crt	STATE_CRITSEC	4	Blocked by another CritSec thread (after attempting to run)
run	STATE_RUNNING	5	Thread currently running
tsd	STATE_TSD	7	Thread waiting for the TSD daemon to page in the TSD.
bst	STATE_READYBOOST	6	Current state never set to this value - see note below.
dly	STATE_DELAYED	8	Delayed TKWakeup (Almost Ready)
frz	STATE_FROZEN	9	Frozen Thread via DosCreateThread, DosExecPgm or DosSystemService
gsk	STATE_GETSTACK	10	TSD daemon is waiting for the page manager to page in a TSD
bad	STATE_BADSTACK	11	TSD failed to page in

Notes: **STATE_READYBOOST** is a modified ready state and never becomes the current state, instead a priority boost is applied and the state becomes **STATE_READY**.

See the **Sta** field description of the **.**P command for related information on thread states.

Pri Thread priority in **TCBPriority** (**TCB** +0x168)

This is the current priority calculated by the scheduler based upon priority class (**TCBPriClass** (**TCB** +0x164)), priority class level (**TCBPrilevel** (**TCB** +0x165)) and priority boosts (**TCBPriClassMod** (**TCB** +0x166)). See **Pri** field description of the .P command for further information.

pTCB

Linear address of the TCB control block that represents the thread.

PriNextQ

The TCB address of the thread at the head of the next priority queue.

PriNextQ is taken from the **TCBpTCBPriNextQ** (**TCB** + 0x170) double-word field.

If there are no other linked priority queues then **TCBpTCBPriNextQ** and **TCBpTCBPriPrevQ** point to this thread and **PriNextQ** and **PriPrevQ** are shown blank.

All TCBs not heading a priority queue have **TCBpTCBPriNextQ** and **TCBpTCBPriPrevQ** pointing to themselves.

PriNext and **PriPrev** is only of relevance to blocked and delayed threads.

PriPrevQ

The TCB address of the thread at the head of the previous priority queue.

PriPrevQ is taken from the **TCBpTCBPriPrevQ** (**TCB** + 0x174) double-word field.

If there are no other linked priority queues then **TCBpTCBPriNextQ** and **TCBpTCBPriPrevQ** point to this thread and **PriNextQ** and **PriPrevQ** are shown blank.

All TCBs not heading a priority queue have **TCBpTCBPriNextQ** and **TCBpTCBPriPrevQ** pointing to themselves.

PriNext and **PriPrev** is only of relevance to blocked and delayed threads.

PriHigh
The TCB address of the next higher priority thread within this priority queue.

PriHigh is taken from the **TCBpTCBPriHigher** (**TCB** + 0x178) double-word field.

If there are no higher priority threads on this priority queue then **TCBpTCBPriHigher** points to this thread and **PriHigh** is shown blank.

PriLow
The TCB address of the next lower priority thread within this priority queue.

PriLow is taken from the **TCBpTCBPriLower** (**TCB** + 0x17c) double-word field.

If there are no lower priority threads on this priority queue then **TCBpTCBPriLower** points to this thread and **PriLow** is shown blank.

PriNext
The TCB address of the next thread of the same priority within this priority queue.

PriNext is taken from the **TCBpTCBPriNext** (**TCB** + 0x180) double-word field.

If there are no other threads of the same priority on this priority queue then **TCBpTCBPriNext** and **TCBpTCBPriPrev** point to this thread and **PriNext** and **PriPrev** are shown blank.

PriPrev
The TCB address of the previous thread of the same priority within this priority queue.

Priprev is taken from the **TCBpTCBPriPrev** (**TCB** + 0x184) double-word field.

If there are no other threads of the same priority on this priority queue then **TCBpTCBPriNext** and **TCBpTCBPriPrev** point to this thread and **PriNext** and **PriPrev** are shown blank.

3.4.22.1 Scheduler Priority Queues

Threads are linked in structures call **Priority Queues** or **PriQ**s.

Priority queues are a double-linked list of thread priority groups. Each group is a double-linked list of threads of the same priority.

Six chain pointers are used for the links of a **PriQ**:

TCBpTCBPriHigher (**TCB** + 0x178)

TCBpTCBPriLower (**TCB** + 0x17c)

TCBpTCBPriNext (**TCB** + 0x180)

TCBpTCBPriPrev (**TCB** + 0x184)

By default these chain pointers are set to point to their own TCB.

TCBpTCBPriHigher and **TCBpTCBPriLower** link the heads of each priority group.

TCBpTCBNext and **TCBpTCBPrev** link the TCBs within each priority group.

A number of **PriQs** are defined. Each is anchored from a global symbol pointer:

_ptcbPriQTSD
Anchor for all threads in **tsd** state.

_ptcbPriQRunner
Anchor for all threads in **rdy** state. At most this contains one TCB.

_ptcbPriQReady
Anchor for all threads in **rdy** state.

_ptcbPriQGetStack
Anchor for all threads in **gsk** state.

_ptcbPriQBadStack
Anchor for all threads in **bad** state.

ptda_pTCBPriQCritSec
Anchor per-process for all threads within a process in **crt** state.

Notes: For the current process **ptda_pTCBPriQCritSec** (**PTDA** +0x2e4) is a also a global symbol. Out of current context it can be located relative to the process' **PTDA** address.

The **TCB** address of the thread that has entered critical section is saved in **ptda_pTCBCritSec** (**PTDA** +0x2e0).

Sleeping threads are queued on priority queues but in a manner to favor wake-up processing. The BlockID is hashed to form an index into a table of **PriQ** anchors. The table is located at global symbol:

_aptcbSleep

Each anchor points to a chain of **PriQs** of threads sleeping on the same BlockID. The head TCB of each **PriQ** within a hashed chain is doubly linked from:

TCBpTCBPriNextQ (**TCB** + 0x170)

TCBpTCBPriPrevQ (**TCB** + 0x174)

Threads that happen to sleep on the same BlockID as a multi-wake-up BlockID are guaranteed not to be put in the same chain as the multi-wake-up threads.

When multi-wake-up threads wake their entire sleeping **PriQ** is moved to a chain of **PriQs** for threads in **dly** state. The delayed thread **PriQ** is anchored from global symbol:

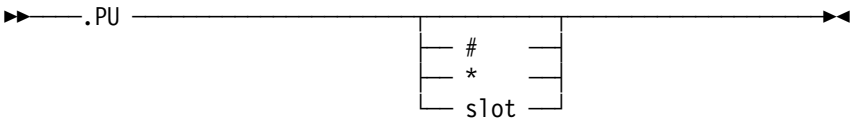
_ptcbPriQDelayed

Since **ptcbPriQDelayed** anchors a chain of **PriQs**, the head of each **PriQ** is doubly-linked using **TCBpTCBPriQNextQ** and **TCBpTCBPriQPrevQ**.

3.4.23 .PU - Display Thread User Space Information

Display thread user space summary information for all (active) threads.

Syntax:



Parameters:

slot

Display user information for thread slot ***slot***.

The following shorthand may be used for the slot number:

***** The current (last) thread the dispatcher gave control to. This value is taken from the word a global label:

```
_TaskNumber
```

The debugger default thread slot. This defaults to the current slot unless overridden by the **.S** command.

If no slot number is given then all thread slots are displayed in slot number order.

Results and Notes:

The **.PU** command locates each thread's TCB from the thread slot table, the linear address of which is given by global variable:

_papTCBSlots

or by traversing the process tree using **TCBpTCBNext** (**TCB** +0x14), **TCBpPTDA** (**TCB** +0x08) and **ptda_pTCBHead** (**PTDA** + 0x20) fields.

Output from the **.PU** command appears in tabular form as follows:

```
 Slot  Pid  Ord  pPTDA    Name     pstkframe  CS:EIP        SS:ESP     cbargs
 0001  0001 0001 ffe3c7d4 *ager    ffe3bf54 1e30:00001794 0030:0000a402 0000
 0002  0001 0002 ffe3c7d4 *tsd
 0003  0001 0003 ffe3c7d4 *ctxh
 0004  0001 0004 ffe3c7d4 *kdb
 0005  0001 0005 ffe3c7d4 *lazyw
 0006  0001 0006 ffe3c7d4 *asyncr
 0008  0002 0001 7b9e4020 pmshell  7b7d7f4c d02f:0000272d 001f:0003f8b8 0008
*000a  0002 0002 7b9e4020 pmshell  7b7dbf44 d087:00003413 bfff:000007a6 0010
 000b# 0002 0003 7b9e4020 pmshell  7b7ddf48 d087:0000351a bfff:00000fc0 000c
```

Each of the fields has the following meaning:

slot

The unique (hexadecimal) index in to the thread slot table of all threads.

This value may be flagged with:

***** To the left to signify the current (or last) dispatched thread.
# To the right to signify the Kernel Debugger or Dump Formatter default thread slot.

Slot may be found in the **TCBNumber** (**TCB** + 0x2) field of the **TCB**.

Pid

The process id this thread slot is assigned to.

Ord

The thread ordinal for this thread slot. This is the unique thread id assigned to the thread within the process to which it belongs.

Ord is taken from the **TCBOrdinal** (**TCB**+ 0x0) field of the current TCB.

pPTDA

Linear address of the PTDA control block representing the process to which this thread belongs. The address is taken from **TCBpPTDA** (**TCB** +0x08).

Name

The name of the primary executable running in this process.

See **name** field description of the **.**P command for further information.

pstkframe

The address of the ring 0 stack frame that saved the user (ring 2 or ring 3) registers at the last transition to ring 0. For internal threads that have never run in ring 2 or ring 3 or for the currently running ring 3 thread this field will appear blank.

The address for the user stack frame is taken from **TCB_pFrameBase** (**TCB** + 0x3c). See **.**R command for further information on displaying registers saved in the user stack frame.

CS:EIP

The user (ring 2 or ring 3) CS:EIP saved in the ring 0 user stack frame the last time the thread made a transition to ring 0. This field will appear blank if the thread is an internal ring 0 thread, currently running in ring 3 or the TSD for this thread is paged out. See the **.**I command for information on paging in a TSD.

SS:ESP

The user (ring 2 or ring 3) SS:ESP saved in the ring 0 user stack frame the last time the thread made a transition to ring 0. This field will appear blank if the thread is an internal ring 0 thread, currently running in ring 3 or the TSD for this thread is paged out. See the **.**I command for information on paging in a TSD.

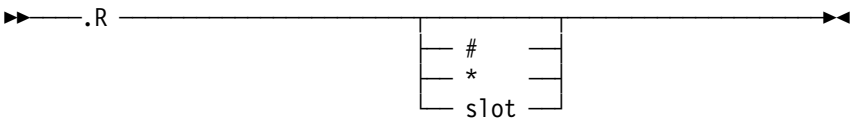
cbargs

The user (ring 2 or ring 3) cbargs saved in the ring 0 user stack frame the last time the thread made a transition to ring 0. This field will appear blank if the thread is an internal ring 0 thread, currently running in ring 3 or the TSD for this thread is paged out. See the **.**I command for information on paging in a TSD.

3.4.24 .R - Display User's Registers

Display the user registers for a given thread slot. Set default addresses for the E, D, K and U commands.

Syntax:



Parameters:

slot

Display user registers for thread slot ***slot***. This option is valid *only* under the Kernel Debugger.

The following shorthand may be used for the slot number:

***** The current (last) thread the dispatcher gave control to. This value is taken from the word a global label:

```
_TaskNumber
```

\# The debugger default thread slot. This defaults to the current slot unless current slot unless overridden by the **.**S command.

If no slot number is given then the debugger's default slot number is assumed.

Results and Notes:

Registers are displayed and register mnemonics are assigned the values displayed for use in address expressions and operands of other Kernel Debugger and Dump Formatter commands.

The register information is obtained as follows:

Under the Kernel Debugger, if the displayed slot is the current system slot and the system is not in kernel mode (that is, **Indos**¬ =1) then the hardware register values save by the debugger are displayed.

Otherwise the registers are extracted from the from the ring 0 stack frame base pointed to by **TCB_pFrameBase** (**TCB** + 0x3c) for the thread slot in question.

The ring 0 stack frame base is created when the threads makes a transition from ring 2 or 3 to ring 0. This happens for a variety of reasons, such as issuing a call gate, trapping, pre-emption, interrupt, etc.. The format of the stack frame base depends on the reason for the ring 0 transition. **TCB_pcriFrameType** (**TCB** + 0x38) points to the CRI, which contains a table of RIPs. Each **RIP** entry is associated with a specific hardware register. The **RIP** contains the offset and length of the associated register saved in the stack frame base. See the Client Register Information and Stack Frames for details of the **CRI** and **RIP** formats.

Note:

If the thread has never run out of kernel mode, as is the case with some internal threads, then the **CRI** is never updated. The **.**R command is not able to format the user registers. For these threads the R command should be used, but only when the thread in question is the current system thread. Because the R command is an alias to the **.**R under the Dump Formatter, there is no way the display the current registers for an internal thread under the Dump Formatter. The only recourse is to display the **TSD** for the thread and attempt to unravel the stack manually.

If an invalid thread slot number is given the Kernel Debugger issues the following message: prompted with the command prompt.

```
Invalid task number: nnnn
```

The format of the **.**R command output depends on whether the RT command has been used to toggle register display to full or short form and also whether the Y 386ENV command has been used to toggle register interpretation into 286 or 386 mode. Examples of the various forms follow:

```
##rt
##.r 2c
eax=f110099f ebx=00000001 ecx=0133fe4c edx=00000007 esi=0133ffec edi=00000000
eip=00000626 esp=0133fe20 ebp=0133fe88 iopl=2 -- -- -- nv up ei ng nz na pe nc
cs=d02f ss=099f ds=0053 es=0053 fs=150b gs=0000  cr2=1581928c  cr3=001d0000
gdtr=7c3e5000 1fff  idtr=ffe00df0 03ff  tr=0010 ldtr=0028 cr0=pg et ts em mp --
dr0=00000000 --e1-  dr1=00000000 --e1-  dr2=00000000 --e1-  dr3=00000000 --e1-
tr6=00000 v=0 d=00 u=00 w=00 c=w  tr7=00000 ht=0 rep=0  dr6=-- -- --  dr7=-- --
002c|d02f:00000626 66ead77a021a5b00 jmp     005b:1a027ad7

##rt
##.r
eax=00000000 ebx=00000014 ecx=0000abd7 edx=0000abd7 esi=00080bff edi=00080007
eip=0000272d esp=0000a668 ebp=0008a670 iopl=2 -- -- -- nv up ei ng nz na pe nc
cs=d02f ss=0047 ds=abd7 es=d137 fs=150b gs=0000  cr2=1581928c  cr3=001d0000
doscall1:CONFORM16:postDOSSEMWAIT:
d02f:0000272d c9              leave


##y 386env
##.r 2c
ax=099f  bx=0001  cx=fe4c  dx=0007  sp=fe20  bp=fe88  si=ffec  di=0000
ip=0626  cs=d02f  ds=0053  es=0053  ss=099f  -- nv up ei ng nz na pe nc
002c|d02f:0626 66ead77a021a5b00 jmp     005b:1a027ad7
##


##rt
##.r 2c
ax=099f  bx=0001  cx=fe4c  dx=0007  sp=fe20  bp=fe88  si=ffec  di=0000
ip=0626  cs=d02f  ds=0053  es=0053  ss=099f  -- nv up ei ng nz na pe nc
gdtr=3e5000 1fff  idtr=e00df0 03ff  tr=0010  ldtr=0028 iopl=2 msw=ts em mp
002c|d02f:0626 66ead77a021a5b00 jmp     005b:1a027ad7
##
```

Following the formatted register display, one line of disassembled code is displayed at the default disassembly address. See the U command for details on disassembling code.

Each of the fields has the following meaning:

General Registers

These comprise the following registers:

ax, bc, cx, dx, sp, bp, si, di

eax, ebx, ecx, edx, esp, ebp, esi, edi

Each is displayed with its value in hexadecimal.

Segment Registers

These comprise the following registers:

cs, ds, es, fs, gs, ss

Each is displayed with its selector value in hexadecimal.

Instruction Pointers

These comprise the following registers:

ip and eip

Each is displayed with its value in hexadecimal.

Flag registers

These comprise the following registers:

flags and eflags

These have their bit setting interpreted as follows:

Bit	Value	Flag	Description
17	1	VM	Virtual 8086 Mode (EFLAGS only)
16	0	RF	Resume Flag - Disable Debug Exceptions (EFLAGS only)
14	1	NT	Nested Task
11	1	OV	Overflow
11	1	NV	¬ Overflow
10	1	DN	Direction Down
10	0	UP	Direction Up
9	1	EI	Enable Interrupts
9	0	EI	Disable Interrupts
7	1	NG	Negative Sign
7	0	PL	Plus Sign
6	1	ZR	Zero Result
6	0	NZ	Non-zero Result
4	1	AC	Auxiliary Carry
4	0	NA	¬ Auxiliary Carry
2	1	PE	Parity Even
2	0	PO	Parity Odd
0	1	CY	Carry
0	0	NC	¬ Carry

Bits 12 and 13 are the I/O Privilege Level bits. These are formatted as **iopl=*level***.

Flags 14, 16 and 17 when reset are indicated by **--**.

Memory Management Registers

gdtr=*xxxxxxxx yyyy*

Global Descriptor Table Register base address (***xxxxxxxx***) and limit (***yyyy***)

idtr=*xxxxxxxx yyyy*

Interrupt Descriptor Table Register base address (***xxxxxxxx***) and limit (***yyyy***)

ldtr=*xxxx*
Local Descriptor Table Register GDT selector (***xxxx***).

tr=*xxxx*
Task Register GDT selector (***xxxx***).

Control Registers

cr0=
System control flags and Machine Status Word.

These have their bit setting interpreted as follows:

Bit	Value	Flag	Description
31	1	PG	Paging Enabled
4	1	ET	Extension Type Flag - x87 support
3	1	TS	Task Switch Flag
2	1	EM	Emulation exception
1	1	MP	Math Present
0	1	PM	Protect Mode Enabled

Reset flag bit are shown with **--**.

cr2=
Page fault linear address.

cr3=
Page Directory Base Register (**PDBR**).

Debug Registers

dr0 to dr3
These are formatted as follows:

dr0=llllllll glxnb
dr1=llllllll glxnb
dr2=llllllll glxnb
dr3=llllllll glxnb

where ***llllllll*** is the breakpoint linear address and ***glxnb*** are **dr7** and **dr6** related flags.

The flags have the following interpretations:

g **G** Indicates a globally enabled breakpoint.

l **L** Indicates a locally enabled breakpoint.

x **E** Indicates an execute breakpoint

R Indicates a read breakpoint

W Indicates a write breakpoint

n The number of bytes tested (1, 2 or 4)

b **B** Indicates a that a debug exception was generated that matched this breakpoint; this is the **B*n*** value of **dr6**.

- Indicates a flag bit reset.

dr6=

The control bits 13-15 are interpreted as follows:

Bit	Value	Flag	Description
15	1	BT	Breakpoint triggered on task switch
14	1	BS	Breakpoint triggered on single step.
13	1	BD	Breakpoint on debug register access/update.

Flag bits not set are indicated by **--**

dr7=

The control bits 8 and 9 are interpreted as follows:

Bit	Value	Flag	Description
9	1	GE	Exact data matching enabled for global breakpoints
8	1	LE	Exact data matching matching for local breakpoints

Flag bits not set are indicated by **--**

Test Registers

tr6=*lllll* v=*v* d=*dd* u=*uu* w=*ww* c=*c*

lllll is the linear page address.

v is **tr6** flag bit 11, the **valid bit.**

dd are **tr6** flag bits 10 and 9.

uu are **tr6** flag bits 8 and 7.

w are **tr6** flag bits 6 and 5.

c is set as follows:

r **tr6** flag bit 0 set. TLB read command.

w **tr6** flag bit 1 reset. TLB write command.

tr7=*ppppp* ht=*h* rep=*r*

ppppp is the **tr7** physical frame address.

h is flag bit 4 value. This is the **hit** or **PL** bit.

r are **tr7** flag bits 3 and 2. These are the **report** or **REP** bits.

The following publications should be consulted for definitive information on processor registers:

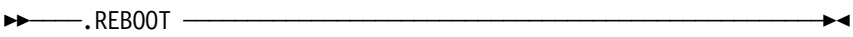
Intel486(TM) Microprocessor Family Programmer's Reference Manual

Pentium(TM) Processor User's Manual

3.4.25 .REBOOT - Restart the System

Restart the system.

Syntax:



Parameters:

None

Results and Notes:

DosHlp service **DosHlpReboot** is called to restart the system.

Attention

No system shutdown processing, whatsoever, is performed.

This command is not available to the Dump Formatter.

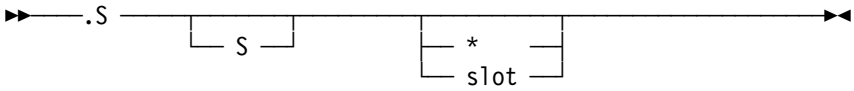
3.4.26 .S - Set or Display Default Thread Slot

Set or display the Kernel Debugger's and Dump Formatter's default slot threads slot.

This command affects the default operation of the following:

D command
E command
U command
.I command
.K command
.P command
.PB command
.PQ command
.PU command
.R command

Syntax:



Parameters:

slot
Set the default threads slot to ***slot***.

The following shorthand may be used for the slot number:

***** The current (last) thread the dispatcher gave control to. This value is taken from the word a global label:

_TaskNumber

If no slot number is given .S displays the current thread slot number in message:

Current task number**:** **nnnn**

where ***nnnn*** is the thread slot number.

S Set current ESP, EBP, SS, CS and EIP registers to those of the Dispatcher.

This option sets these registers as if the thread context had just been switched by the OS/2 Dispatcher. The R command will show the thread in kernel mode, about to be run.

No actual updating of register values takes place. Only default values are effected.

The new defaults are derived as follows:

ESP taken from **TSDKernelESP** (TSD + **Disp** value of .P command output.)

EBP taken from **TSDUserSSPad** (TSDKernelESP - 2)

SS selector 30 (**TASKAREA** segment).

CS Selector 170 (**DOSHIGH32CODE** segment).

EIP label **pgSwitchRet**.

This option is not available to the Dump Formatter.

It is *not* generally of relevance to non-kernel mode code.

Results and Notes:

The **.**S command sets certain default values such that the view of the user's space in the new default slot is as if the thread context had switched. Linear and LDT selector based addresses will be accessed correctly by the Dump Formatter and Kernel Debugger. However, certain system data that are updated by a context switch are not changed and continue to display in the system's current thread context. These items include:

Task Register (TR)

GDT descriptor table entries for selectors 28, 30, 38 and 150b

Current TSS ring 0, and ring 2 stack selectors and pointers

Global and System copy of the Current Local Information Segments

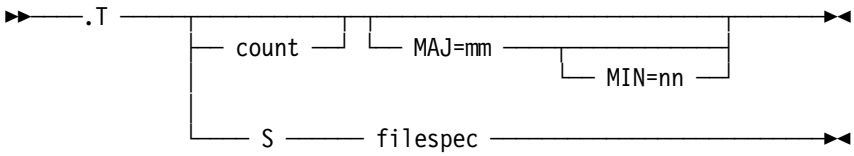
The Thread Local Memory Area and Local Information Segment mapped by LDT descriptor **dfff**

Note: Descriptor **dfff** maps a global shared memory object, but it's data is copied from the incoming **PTDA** and **TCB** when a context switch occurs. This achieves the effect of thread local memory.

3.4.27 .T - Dump the System Trace Buffer

Dumps the system trace buffer.

Syntax:



Parameters:

count
The number of trace entries to print, starting with the most recent. If not specified then the entire trace buffer will be dumped.

MAJ=*mm*
Specifies that only trace events with major code ***mm*** should be displayed.

See System Trace Facility - Major Code Assignments for a information on the deployment of trace major and minor codes in OS/2.

> **Attention**
>
> The Kernel Debugger may fail to process the **MAJ=** parameter correctly. Under some circumstances the debug kernel may hang. Use this option advisedly!

MIN=*nn*
Specifies that only trace events with minor code ***nn*** should be displayed.

This option required the specification of a major code using the **MAJ=** parameter.

See System Trace Facility - Major Code Assignments for a information on the deployment of trace major and minor codes in OS/2.

> **Attention**
>
> The Kernel Debugger may fail to process the **MAJ=** parameter correctly. Under some circumstances the debug kernel may hang. Use this option advisedly!

S Specifies that the trace buffer should be saved to a file named in ***filespec***.

This option is *only* available to the Dump Formatter.

The saved trace file may be subsequently formatted using the OS/2 TRACFMT command.

filespec
The file specification for the saved trace buffer.

The ***filespec*** may be fully qualified. The path defaults to the current directory.

Results and Notes:

The trace is activated using the OS/2 TRACE command.

If the trace is not active then the following message is generated:

```
Trace not on
```

The trace buffer is allocated in a single segment (STDA) whose selector may be located from global symbol **ras_stda_addr**. The STDA is a circular buffer whose entries are recorded in reverse order. The header gives the offsets to the first, last and current entires. The format of the trace buffer is described under System Trace Data Area.

The major codes being traced are recorded in a bit string located at label **ras_mec_table**. Each active major code has its corresponding bit set.

The minor codes being traced are recorded in a bit string whose selector is located at label **ras_min_table**. The minor code table contains 32 byte entries, each corresponding to a major code. Each bit of each entry corresponds to a minor code within the major. If the bit is set, then the minor code is traced.

When tracing by Pid is active then the **ptda_rasflag** (**PTDA** +0x39a) is set to **0xff**.

The status of system tracing is recorded in status byte at label **ras_systr_flags**. The following flags are defined:

Name	Bit mask	Description
RF_TRCAVAIL	0x80	System Trace Available
RF_TRCPAUSED	0x40	Trace paused
RF_TRCPID	0x20	Trace by PID
RF_TRCERRCOUNT	0x10	Tracing until error count
RF_TRCSUSPEND	0x08	Suspend due to error count
RF_TRCMINORCD	0x04	Tracing by Minor Code

The **.T** command output appears as follows, (see note at the end of this section for information on recent changes to the format of the trace output):

```
MAJ=04  MIN=0089  PID=0006  CONTEXT=KERNEL:PROTECT
MAJ=06  MIN=008c  PID=0000  CONTEXT=KERNEL:PROTECT
   00 00                                          ..
MAJ=06  MIN=000c  PID=0000  CONTEXT=KERNEL:PROTECT  TS=1336
   08 00                                          ..
MAJ=04  MIN=0009  PID=0006  CONTEXT=KERNEL:PROTECT
MAJ=04  MIN=0089  PID=0006  CONTEXT=KERNEL:PROTECT
```

Each of the fields is defined as follows:

MAJ=
The traced event major code.

MIN=
The traced event minor code.

PID=

The current Pid when the event occurred. See **.**P command for information on displaying active Pids.

CONTEXT=*system:processor*

The system and processor context under which the event was traced.

system context may be:

KERNEL If the trace record was created internally by a kernel routine.

API If the trace record was created externally by the use of the **DosDynamicTrace** or **DosSysTrace** APIs.

See Dynamic Trace Customizer for information on creating dynamic trace records (via **DosDynamicTrace**).

See DosSysTrace (Static Trace Event Recording) for information on creating static trace records.

processor context may be:

PROTECT If the trace record was created when the system was running in protect mode.

REAL If the trace record was created when the system was running in real mode.

TS=*hhss*

The system time stamp where ***hh*** is 100th seconds and ***ss*** is seconds.

The time stamp is taken from the Global Information Segment **(GISEG+0xa).** It is only recorded in the trace record if the time has changed since the previous timed stamped record was recorded.

Note: **TRACEFMT** treats this value as a word length fixed number of two decimal places.

trace data

Additional trace data.

A trace event may be accompanied with additional trace data, in which case it is dumped in hexadecimal and ASCII format on the following line.

Related information on the system trace facility may be found in:

System Trace Reference Manual, of The OS/2 Debugging Library.

The OS/2 Command Reference, TRACE command.

The OS/2 Command Reference, TRACEFMT command.

The OS/2 Command Reference, TRACEBUF CONFIG.SYS statement.

3.4.27.1 New Trace Format

From OS/2 2.11 fix pack 91 and OS/2 3.0 fix pack 8, the system trace has been enhanced to include more useful timestamp information. The Kernel Debugger and Dump Formatter were updated in FixPaks 16 (OS/2 3.0) and 105 (OS/2 2.11) to take account of the new format.

Attention

The use of the **.T** command after the new trace format was implemented, but before the Kernel Debugger and Dump Formatter were updated, caused the Kernel Debugger and Dump Formatter to trap.

The following is an example of the new format:

```
#
Trace On at 0000,0000,0000,0000,0000,0000,0000
Trace Off at 0000,0000,0000,0000,0000,0000,0000
  MAJ=03  MIN=0009  PID=0000  CONTEXT=KERNEL:PROTECT  TS=3611,382e
     00 00 00 00 bd 55 f5 ff 60 01 00 00 02 00 01 00   ....=Uu.`.......
  MAJ=03  MIN=000f  PID=0000  CONTEXT=KERNEL:PROTECT  TS=3611,382e
     00 00 cc cc f1 27 00 00 00 10 00 00 06 02 01 00   ..LLq'..........
     c8 3c f2 ab                                       H<r+
  MAJ=03  MIN=0008  PID=0017  CONTEXT=KERNEL:PROTECT  TS=3611,252d
     00 00 00 00 93 86 e5 1b 5b 00 00 00 02 22 01 00   ......e.[....".
  MAJ=03  MIN=0008  PID=0017  CONTEXT=KERNEL:PROTECT  TS=3611,222d
     00 00 00 00 93 86 e5 1b 5b 00 00 00 02 22 01 00   ......e.[....".
  MAJ=03  MIN=0008  PID=0017  CONTEXT=KERNEL:PROTECT  TS=3611,222d
     00 00 00 00 93 86 e5 1b 5b 00 00 00 02 22 01 00   ......e.[....".
```

The formatted trace is headed by a pair of timestamps that give the time tracing was initiated and terminated. These are of the form:

```
YYYY,xxMM,xxDD,xxHH,xxmm,xxss,xxhh
```

Where:

YYYY is years,

MM is Months

DD is Days,

HH is hours,

mm is minutes,

ss is seconds.

hh is 1/100th seconds,

xx ignore.

The timestamp of each trace record is now shown as a pair of word values of the form:

```
TS=MMHH,hhss
```

Where

MM is minutes,

HH hours,

hh 1/100s seconds and

ss seconds.

Note:

The byte reversal occurs because the time values are originally byte values which are displayed as words.

Glossary

Application Anchor Block (AAB). A PM **Application Anchor Block** is allocated in the Thread Local Memory Area (TLMA) when a PM application thread creates a message queue. The AAB contains a pointer to the MQ which allows PM to find the MQ in any context. This is particularly useful to the debugger since it also allows the MQ of any PM thread in the system since the TLMA is saved in a thread′s TCB.

BlockIDs. BlockIDs are conventional tokens used to represent the reason for a thread that blocks. This occurs as the result of the kernel entering TKSleep (either directly or via ProcBlock). The address of the BlockID is passed to TKSleep and stored in TCBSleepID. A thread wakes when the kernel calls TKWakeup (or ProcRun) with a corresponding BlockID. All, zero or the highest priority thread blocked on the BlockID will be woken depending on parameter flags. This mechanism is used by most functions and APIs that cause thread execution to be suspended, either for an event or serialisation.

Examples are:

DosSleep

DosSemWait

DosWaitChild

DosRead

DevHlp_ProcBlock

Refer to 3.4.21, ".PB - Display Blocked Thread Information" on page 246 for more detailed information.

BIOS Parameter Block. A BIOS Parameter Block is used for low level Disk I/O calls to the BIOS.

For further information see:

The .D BPB Command in the Kernel Debugger and Dump Formatter Command Reference.

The BPB Structure in the System Reference.

Breakpoint. A breakpoint is a location in a program where execution is suspended and control is given to a debugging tool.

The INTEL architecture supports two implementations of breakpoints for debugging purposes:

The software generated breakpoint using the INT 3 instruction;

The hardware generated breakpoint using the Debugging Registers.

The use of software breakpoints require code modification, whereas the use of debugging registers does not. However, the number of predefined software breakpoints is potentially unlimited whereas there are only 4 breakpoints specifiable using Debugging Registers.

A further distinction between the two types is that software breakpoints only intercept the execution of a particular instruction path, whereas Debugging Registers may be used, in addition, to intercept data fetches and stores from a particular location in virtual memory.

The Kernel Debugger supports both implementations of breakpoints through the use of the:

The BR command, which uses Debugging Registers.

The BP command, which uses INT 3 instructions.

The Kernel Debugger limits the predefinition of BP breakpoints to 10, however the programmer may code as many additional INT 3 instructions into thier program as desired.

The Kernel Debugger refers to breakpoints explicitly set by the BP and BR commands as ***sticky*** (implying a certain permanence about them). The G command may have one or more temporary breakpoints established when one or more stop addresses are specified. These are referred to as ***go*** breakpoints. Once the Kernel Debugger breaks in ***go*** breakpoints are removed. The internal operation of the Kernel Debugger may also necessitate the use of the occasional temporary breakpoints when instruction tracing (see the T and P commands). These are set implicitly and discarded without the user being aware of their existence. Go and temporary breakpoints are created using the INT 3 instruction. Go and sticky BP breakpoints count towards the Kernel Debugger imposed limit of 10, but temporary breakpoints only ever exist singly so do not.

Block Management Package (BMP). A Block Management Package (BMP) is a data structure used to manage a pool of fixed length blocks using a bit string. Each bit in the bit string corresponds to an entry. A set bit indicates whether the entry is in use.

Typically this is used for:

Kernel Heap allocation
Memory object allocation

cbargs. cbargs is the argument count associated with the hardware defined call gate mechanism. The count is the number of words or double-words (as defined by the gate descriptor) that are inserted into a ring 0 stack when ring 2 or ring 3 code executes a call gate instruction.

CBIOS. The Compatibility BIOS (CBIOS) is a layer of code in the OS2LDR that presents a hardware independent interface to the BIOS for the OS2KRNL. The interface to the OS2KRNL is provided through a set of entry points called Dos Helper Functions.

Client Register Information (CRI). The Client Register Information (CRI) is a table of Register Information Packets (RIPs) that describe the offset and length of each register that is stored in a ring 0 stack frame on entry to the kernel. This level of indirection allows kernel routines to access entry registers regardless of the stack frame type, of which there are a number, for example:

System Entry Frames from API calls

Trap Frames from traps and exceptions

Interrupt Frames from the interrupt manager

VDM Stack Frames

Kernel Stack Frames

Each TCB points to a CRI and the associated stack frame from TCB_pcriFrameType(TCB + 0x38) and TCB_pFrameBase(TCB + 0x3c) respectively.

Codepage Data Information Block (CDIB). The Codepage Data Information Block (CDIB) contains country-specific constant information relating to screen, keyboard and printer devices. The CDIB is built from information derived directly from CONFIG.SYS statements.

The CDIB may be located from the SAS.

Command Subtree Identifier. The Command Subtree Identifier is used to represent a part of a process (or command) tree headed by a particular parent process. The ID used is the Pid of the process that heads the subtree.

Normally a process has a CSID equal to it's own Pid. However, when processes become orphaned they acquire the subtree Id of their original parent and become adopted by their grand-parents by acquiring their grand-parents's Pid as their new parent Pid.

Common ABIOS Data Area (CDA). The CDA is the Common ABIOS Data Area.

Refer to the Kernel Debug and Dump formatter guide, external command .C - display Common ABIOS data area.

Compatibility Region Mapping Algorithm. The Compatibility Region Mapping Algorithm (also referred to as the thunking algorithm) is used by thunking code to convert 16:16 addresses to 0:32 addresses and vice versa.

This is achieved by ensuring LDT selectors have their limits set to 64K so that they tile the compatibility region (0M to 448M). This gives an easy conversion algorithm from the selector:offset address to the 32-bit linear address. In C language syntax this is expressed as follows:

```
linear_address=((selector >> 3) << 16) + offset

selector:offset=((linear_address. >> 13) | 7):(linear_address & 0x0000ffff)
```

Context. **Context** (or thread context) refers to the *view* of the system any given thread has. Only one thread context may be current at any time.

Switching contexts refers to the process of preparing the system for another thread to run. From an application program′s perspective this implies restoring its registers and ring 2 and 3 stacks when it is given the opportunity to run again. From the system′s perspective, restoration of an application′s registers and stacks is done after the context switch, by the dispatcher, on exiting kernel mode. Not every context switch is followed by exiting kernel mode. For example, if another thread in the same process is in critical section (but blocked) then the new thread enters **crt** state and the scheduler is called to select yet another thread.

Context switching includes the following system actions:

- Updating GDT descriptor entries 28, 30, 38 and 150b, which point to

  The current process′ **LDT**,

  The current threads ring 0 stack,

  The current thread′s floating point emulator work area,

  The current thread′s TIB. (By default the **FS** selector is loaded with 150b).

  Note: The **LDT** selector is only updated when the process changes with a context switch, that is, for a process context switch.

- Updating page directory and tables for a process context switch.
- Updating the **TR** register if the process switch involves a task switch (normally only VDMs).
- Updating the current TSS ring 0 and ring 2 stack addresses.
- Updating system copies of the Global and Information Segments.
- Copying the Local Information Segment from the incoming PTDA and the Thread Local Memory Area from the incoming TCB to the segment mapped by LDT selector **dfff**.

Besides addressing the current ring 0 stack, selector 30 also addresses the current thread′s scheduling control blocks. In particular: the PTDA, TCB and TSD. This is done by aliasing selected address ranges from selector 30 to those of the true PTDA, TCB and TSD in the system arena global memory for the current context. The system defines a dummy module containing a hard-coded PTDA. The symbols of this module have the same name as those of the fields in the PTDA. The system arranges for this to map the PTDA addressed by selector 30. This trick allows the system to refer to PTDA fields for the current context without regard for which process is current, simply by using the field names as public symbols. The user may use the same symbols for referencing the PTDA but these are only valid for the current system context. To access PTDA fields in other contexts the following technique can be used:

```
Note that the current PTDA is located at PTDA_Start

##.p *
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
*0025  0004 0002 0004 0001 blk 0300 7b7c8000 7bbc4080 7bbe8a90 1fc4 16 someprog

The current thread slot is 25

We wish to know the thread that has entered critical section in process of
thread slot 40.  The address of the critical section TCB is
saved in ptda_pTCBCritSec and the thread ordinal and slot number
are the first two words of the TCB.

##.p 40
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
 0040  0012 0002 0012 0001 blk 0500 7b7d6000 7b9e4020 7b9c8a70 1eb8 10 userprog

##dw %(DW(%7b9e4020+ptda_ptcbcritsec-ptda_start)) l2
%7b9c8de0 0002 0041

Thread 2 of 12 or thread slot 41 is in critical section
##.p 41
 Slot  Pid  Ppid Csid Ord  Sta Pri  pTSD     pPTDA    pTCB     Disp SG Name
 0041  0012 0002 0012 0002 blk 0800 7b7da000 7b9e4020 7b9c8de0 1ed4 10 userprog
```

Refer to the Kernel Debugger and Dump Formatter .P and .S commands for more information.

Current Directory Structure (CDS). A Current Directory Structure (CDS) is used to store file system information about the current directory per drive of each process.

Each CDS is managed in an RMP segment. The PTDA for each process contains an imbedded array of 26 CDS handles, one for each drive. The CDS RMP segment may be located from the SAS.

See also related structures:

- MFT
- SFT
- DPB
- FSC
- VPB

Refer to the following for more detailed information

- 3.4.2, ".A - Format the System Anchor Segment (SAS)" on page 149.
- 3.4.5, ".D - Display an OS/2 System Structure" on page 160.

DEM. DEM is the DOS Emulation component of OS/2.

Driver Parameter Block (DPB). A Driver Parameter Block (DPB) contains vital information about the state and format of a disk drive. The DPBs are chained together and located in a single segment whose selector may be obtained from the SAS. See also 3.4.2, ".A - Format the System Anchor Segment (SAS)" on page 149.

See also related structures:

- CDS
- MFT
- SFT
- FSC
- VPB

Refer to the following for more detailed information

- 3.4.2, ".A - Format the System Anchor Segment (SAS)" on page 149.
- 3.4.5, ".D - Display an OS/2 System Structure" on page 160.

DosHlp. DosHlp services comprise a set of hardware dependent service routines established during system initialisation for use by the OS2KRNL and user programs via the OEMHLP$ device driver. Many of the DosHlp services deal with device dependent BIOS behaviour and therefore provide a device independent interface to the BIOS.

File Allocation Table (FAT). The File Allocation Table file system is the default filing system supported by OS/2. Support for FAT is always present, regardless of any installed file systems.

File System Control Block (FSC). A File System Control Block (FSC) represents an installed file system (IFS). The FSC contains a table of entry points implemented by the file system driver (FSD). All FSCs are located in a single segment whose selector may be obtained from the the SAS, see 3.4.2, ".A - Format the System Anchor Segment (SAS)" on page 149.

See also related structures:

CDS

MFT

SFT

DPB

VPB

File System Driver (FSD). A File System Driver (FSD) is a special load module that implements an installed file system (IFS). FSDs are loaded during system initialization when the .IFS statement of CONFIG.SYS is encountered.

Examples of FSDs are:

HPFS.IFS

HPFS386.IFS

CDROM.IFS

Gate. A gate descriptor is one that defines to the hardware a means of entering code that executes at a more privileged level of authority. Four types of gate are defined:

Call Gate	The subject of a CALL instruction. Typically used to implement operating system and device driver application programming interfaces (APIs). Device drivers may create *Call gates* dynamically using the DosDynamicAPI facility.
Task Gate	The subject of a call or exception where a (hardware assisted) task switch is required.
Interrupt Gate	The subject of a hardware or software generated interrupt. Typically an Interrupt gate will switch execution to an interrupt handler when a device presents an interrupt.
Trap Gate	The subject of a trap exception. Used to handle programming errors.

Global Descriptor Table (GDT). The Global Descriptor Table (GDT) is a hardware architected control block. The GDT is common to all protect mode processes. It contains descriptors for memory segments common to all protect mode processes.

Refer to 3.3.17, "DG - Display Global Descriptor Table" on page 97 for more detailed information.

Global Information Segment (GISEG). The Global Information Segment (GISEG) is a single instance control block that records the current session status, date and time, trace status and version of the system.

The system maintains two copies of the Global Information Segment to fence against system damage.

The selector for the GISEG may be located from the SAS. See the Dump Formatter and Kernel Debugger .A command.

The GISEG is also mapped locally per-process by the LDT descriptor 0xdff4.

hal. The memory alias record handle (hal) is an index into the table of memory alias records (VMALs) whose address is located at **_parVMAliases.**

Refer to 3.4.15, ".ML - Format Memory Alias Records (VMAL)" on page 209 for more detailed information.

har. The memory arena record handle (har) is an index into the table of memory arena records (VMARs) whose address is located at **_parvmOne.**

Refer to 3.4.12, ".MA - Format Memory Arena Records (VMAR)" on page 197 for more detailed information.

hco. A hco is a handle for a memory context record. This is an index into the table of memory arena records (VMCOs) whose address is located at **_pcovmOne.**

Refer to 3.4.13, ".MC - Format Memory Context Records (VMCO)" on page 204 for more detailed information.

hmte. The MTE control block handle (hmte) is the hob of the memory object that contains the MTE.

MTEs are allocated as pseudo-objects, so do not have Arena Records associated with them.

Refer to the following for more detailed information:

3.4.16, ".MO - Format Memory Object Records (VMOB)" on page 212.

3.4.10, ".LM - Format Loader Structures (MTE, SMTE, OTE and STE)" on page 190.

hob. The memory object record handle (hob) is an index into the table of memory objects records (VMOBs) whose address is located at **_pobvmOne.**

Refer to 3.4.16, ".MO - Format Memory Object Records (VMOB)" on page 212 for more detailed information.

hptda. The PTDA control block handle hptda is the hob of the memory object that contains the PTDA.

PTDAs are allocated as pseudo-objects, so do not have Arena Records associated with them.

Refer to 3.4.16, ".MO - Format Memory Object Records (VMOB)" on page 212 for more detailed information.

Interrupt Descriptor Table (IDT). The Interrupt descriptor table (IDT) is a hardware architected structure that comprises a table of gate descriptors, one for each interrupt vector. The low numbered entries are defined by the hardware architecture and dedicated to exception management.

The Kernel Debugger's V command may be used to intercept system exception handlers.

Interrupt Vector. An interrupt vector is presented to the processor when an interrupt is generated either externally by the Programmed Interrupt Controller or internally within the processor chip itself. It is used by the processor as an index into the IDT to determine which interrupt routine should be dispatched.

The processor reserves vectors 0 - 31 to correspond to hardware architected exceptions 0 through 31. Vectors 32 - 255 are reserved for I/O interrupts, which are presented to the processor by the Programmed Interrupt Controller when the one of its IRQ lines is triggered. The correspondence between vectors and IRQs is defined during system initialisation as follows:

IRQs 0 - 7 vectors 0x50 - 0x57

IRQs 8 - 15 vectors 0x70 - 0x77

Thus a keyboard interrupt, which is assigned to IRQ 1 under the IBM PC architecture will be handled by the interrupted handler whose interrupt gate is assigned to IDT descriptor 0x51.

See the Dump Formatter and Kernel Debugger DI command for information on displaying IDT entries.

Internal Processing Errors (IPEs). Internal Processing Errors are unrecoveralble error conditions detected by the system while running in ring 0. The may arise from inconsistencies detected by the OS/2 Kernel or from traps occuring in any ring 0 code (Kernel, Installable File System Drivers and Device Drivers).

When the system detects an IPE it enters a routine called panic where an error message is formatted and displayed and the system is halted.

Job File Number (JFN). A Job File Number (JFN) is a handle for open file system objects, unique within the process that opened the file system object. The JFN is returned by DosOpen. It is used and an index into the JFN Table to locate the corresponding SFT handle.

Job File Number Table (JFT). A Job File Number Table (JFT) entry is assigned to each open file system object within a process. The JFT provides a cross-reference to the handle for the corresponding SFT. The JFT is locatable from the PTDA field **JFN_pTable** (**PTDA** +0x5b8 (H/R: +0x5b0)) for each process.

The JFT is initially allocated within the PTDA at label **JFN_Table** (**PTDA** +0x35e) with 20 entries. If this is expanded by use of the DosSetMaxFH then JFN_pTable is updated to point to the new table.

See also related structures:

- CDS
- DPB
- MFT
- FSC
- VPB

Loader Cache. The Loader Cache is used for saving discardable pages of instance data segments from DLLs loaded from mountable media. The caches is allocated from the kernel heap and has a system object owner ID of cache.

Local Descriptor Table (LDT). The Local Descriptor Table (LDT) is a hardware architected table of memory descriptors.

Under OS/2 one LDT is allocated per process.

Refer to 3.3.19, “DL - Display the Current Local Descriptor Table” on page 101 for more detailed information.

Local Information Segment. The Local Information Segment is a per-process control block that records the current status of the process. It is imbedded in the PTDA and is also mapped by the LDT descriptor 0xdfff.

Master File Table (MFT). A Master File Table (MFT) entry is used to associate path names with open files (SFTs) and lock records (RLRs). The MFTs are managed in a PTREE structure, which is locatable from the SAS.

See also related structures:

- CDS
- DPB
- SFT
- FSC
- VPB

Refer to the following for more detailed information:

3.4.2, ".A - Format the System Anchor Segment (SAS)" on page 149

3.4.5, ".D - Display an OS/2 System Structure" on page 160

Message Queue Header (MQ). A PM **Message Queue Header (MQ)** is used as an anchor for message processing for a given PM Application's message thread. The MQ is created when a threads calls **WinCreateMsgQueue**.

Module Table Entry (MTE) (non-swappable). The (non-swappable) Module Table Entry (MTE) for a loaded module is use to record information about loaded modules. Since the MTE is allocated in non-swappable only information that must be resident at all times is recorded here. Related information that may be paged out is recorded in its sister control, the Swappable Module Table Entry (SMTE).

The MTE contains the following information:

pointers to related control blocks such as, SMTE, resource and fix-up tables.

attributes of the load module.

Use count for .EXE modules.

Each MTE is identified by a unique handle referred to as the hmte.

Refer to 3.4.10, ".LM - Format Loader Structures (MTE, SMTE, OTE and STE)" on page 190 for more detailed information.

Object Table Entry (OTE). An Object Table Entry (OTE) describes the address, size and attributes an object within a loaded 32-bit load module.

The corresponding control block for a 16-bit load module is the STE.

Refer to 3.4.10, ".LM - Format Loader Structures (MTE, SMTE, OTE and STE)" on page 190 for more detailed information.

Page Frame Structure (PF). A Page Frame Structure (PF) is used by page frame management to track the status of a physical storage frame. The Page Frame Structures are allocated in contiguous storage, anchored from the address specified in global variable:

_pft

Each PF corresponds one to one with a frame of physical storage and provides links to Virtual Page Structures VPs.

Zero or more PTEs may be pinned to a physical frame, this is reflected in a reference count maintained in the associated PF.

UVIRT mappings have their corresponding PFs reserved unless aliased by non-UVIRT storage.

Refer to the following for more detailed information

3.4.17, ".MP - Format Memory Page Frame Structures (PFs)" on page 225.

Paragraph. A paragraph is a unit of memory allocation of 16 bytes. Paragraph aligned allocations lie on a 16-byte boundary.

Patricia Tree (PTREE). A Patricia Tree (PTREE) is a form of tree structure designed to offer a fast look-up facility for generically specified keys. In OS/2 a modified form of the PTREE is use to manage MFTs for fast path-name look-up.

Page Table Entry (PTE). A Page Table Entry (PTE) is a hardware architected structure that is used to map virtual addresses to physical storage addresses.

Refer to 3.3.20, "DP - Display Page Directory and Table Entries" on page 102 for more detailed information.

Per-Task Data Area (PTDA). The Per-Task Data Area (PTDA) is the anchor point for all process (task) related control information. One PTDA exists per process and from it is located the LDT, TCB chain, Page tables and Arena Headers for a process.

All active PTDAs are addressable, whatever the current process, from a global address in the system arena. However, for the current process an alias address is created using selector 30 and in addition the many of the PTDA field names are declared as public symbols. This allows the fields names in the PTDA for the current process to be referred to directly under the Kernel Debugger and Dump Formatter.

PTDA_Start is the symbol assigned to the beginning of the current PTDA. Using the ? command against this and other PTDA field names allows relative offsets for PTDA fields to be calculated and used in other contexts as offsets from the global PTDA address.

Refer to the following for more detailed information

3.4.20, ".P - Display Process Status" on page 238.

Physical Arena Information block (PAI). The Physical Arena Information block (PAI) describes ranges of physical memory to memory management.

Pageable physical memory is described by the PAI pointed to by the SAS.

Process Information Block (PIB). The Process Information Block (PIB) is a supplemental process related control block made accessible to ring 3 programs. It contains process status information obtained from the process′ PTDA.

The PIB may be located from ptda_avatib(PTDA + 0x28) using the Dump Formatter or Kernel Debugger.

A program gains access to the PIB along with the TIB by calling the DosGetInfoBlocks API.

Process Identifier (pid). The Process Identifier (pid) is a unique system wide value used to identify a given process.

Note: It is not the same as the hptda which also uniquely identifies a process.

The Pid is used as a handle in process related APIs such as DosKillProcess and DosWaitChild.

Refer to 3.4.20, ".P - Display Process Status" on page 238 for more detailed information.

Program Data Block. The Program Data Block is the name given to the DOS PSP by the DEM component of OS/2.

Program Segment Prefix (PSP). The Program Segment Prefix (PSP) is a DOS control block that forms the header of a loaded program. Under OS/2 the DEM component refers to this as the PDB or Program Data Block.

Process. A process is a collection of threads that share a common address space.

Each process is primarily represented by a PTDA structure and is assigned a unique identifier, the Pid.

Processes are organised in hierarchical tree structures known as process or Command Subtrees.

Pseudo-Objects. Pseudo-Objects are small system objects that comprise control blocks and other system areas, which for reasons of virtual memory conservation are not represented by a corresponding Arena Records. They are allocated out of the kernel resident heaps and comprise the following types of object:

MTE

VMAH

PTDA

Loader Cache

Refer to 3.4.16, ".MO - Format Memory Object Records (VMOB)" on page 212 for more detailed information.

Queue Message (QMSG). A PM **Queue Message (QMSG)** is used by **WinPostMsg** to enqueue an asynchronously sent message to a thread's message queue. QMSGs are chained from the MQ of the receiver in a circular array.

Record Lock Record (RLR). A Record Lock Record (RLR) describes a locked region of a file system record. RLRs are chained from the related MFT and point to the associated SFT. They record the owner of the record lock.

See also related structures:

CDS

DPB

SFT

FSC

VPB

Record Management Package (RMP). A Record Management Package (RMP) is a data structure designed for tabulating variable length records. Typically OS/2 uses RMPs to manage:

Named Storage names
Open File names
Directory names
System Semaphore names

Reliability, Availability and Serviceability (RAS). RAS is an acronym that refers to diagnostic and service support within OS/2. Frequently it is used as a synonym for the adjective diagnostic.

Register Information Packet (RIP). A Register Information Packet (RIP) is an entity used to describe the size and offset of a register in a system stack frame. RIPs are located in a CRI.

Scheduler. The Scheduler component of OS/2 is responsible for managing threads on queues according to priority and status.

Refer to the following for more detailed information

3.4.20, ".P - Display Process Status" on page 238

3.4.21, ".PB - Display Blocked Thread Information" on page 246

3.4.23, ".PU - Display Thread User Space Information" on page 256

3.4.22, ".PQ - Display Scheduler Queue Information" on page 251

Screen Group. A Screen Group is a logical full screen buffer and keyboard. A number of processes may be assigned to run in a given screen group. The workplace shell is one such screen group. Each screen group is assigned an ID. The screen group assigned to a process is recorded in its Local Information Segment. The currently active screen group is recorded in the Global Information Segment.

Screen Groups are represented by SGCB structures.

Under version 2 of OS/2 the screen group concept has been extended to that of a session.

Refer to 3.4.20, ".P - Display Process Status" on page 238 for information on displaying screen group ids.

Screen Group Control Block (SGCB). The Screen Group Control Block (SGCB) is used by the session manager component of the system to represent a Screen Group. It contains status information for the screen group and acts as a cross reference between the Pid currently associated with a given screen group and vice versa.

Segment Table Entry (STE). A Segment Table Entry (STE) describes the address, size and attributes of a segment (object) within a loaded 16-bit load module.

The corresponding control block for a 32-bit load module is the OTE.

Refer to 3.4.10, ".LM - Format Loader Structures (MTE, SMTE, OTE and STE)" on page 190 for more detailed information.

Send Message Structure (SMS). A PM **Send Message Structure (SMS)** is used by **WinSendMessage** to enqueue a synchronously sent message. SMSs are chained from the MQ of both the sender and receiver.

Session. Sessions are groups of related processes initiated using DosStartSession API. Each session is assigned a logical screen buffer or presentation space. Sessions are identified by a unique ID that corresponds with their Screen Group Id (though the range of numbers is extended to included PM sessions, which all share then same screen group).

The following session ID/Screen Group ID ranges are defined:

SG	Usage
0	Hard Error Popups
1	Shell Screen Group
2	Real Mode Screen Group
3	VioPopUp Screen Group
4	First Full Screen Application Session
15	Last Full Screen Application Session
16	First Windowable VIO-Session
31	Last Windowable VIO-Session
32	First PM session
255	Last PM session

System Anchor Segment (SAS). The System Anchor Segment (SAS) is a central system control block use to anchor control blocks for major system components such as:

File systems

Device Drivers

Scheduler

Memory management

The SAS is built at the beginning of the segment addressable from selector 70 and 78.

Refer to the following for more detailed information

3.4.2, ".A - Format the System Anchor Segment (SAS)" on page 149.

System Queue Message (SQMSG). A PM **System Queue Message (SQMSG)** is used by the **PMDD.SYS** device driver to enqueue messages, which represent system input activity, to the system input queue.

Swappable Module Table Entry (SMTE). The Swappable Module Table Entry (SMTE) contains characteristics of a loaded module that may be page out of memory. The SMTE is the sister control block to the MTE, which records those characteristics that must be resident at all times.

The SMTE principally contains:

A pointer to OTE or STE.

A pointer to the fully qualified module name.

The entry point and initial stack pointers.

Refer to 3.4.10, ".LM - Format Loader Structures (MTE, SMTE, OTE and STE)" on page 190 for more detailed information.

System File Table (SFT). A System File Table (SFT) entry is used to describe the attributes of each instance of an open file system object. SFTs are stored in a segment directly locatable from the SAS. SFTs are indirectly locatable from the JFN Table imbedded in the PTDA of each process that opens a file system object.

See also related structures:

CDS

DPB

MFT

FSC

VPB

Refer to the following for more detailed information

3.4.2, ".A - Format the System Anchor Segment (SAS)" on page 149

3.4.5, ".D - Display an OS/2 System Structure" on page 160

System Trace Data Area (SDTA). The System Trace Data Area (SDTA) is a circular buffer used to record trace events. The SDTA may be located from the SAS.

Symbol. A symbol is the name given to a program code or data location that has been made public by the programmer. Such symbolic definitions appear in the map file output from the linkage editor. They may be referenced in the Dump Formatter and Kernel Debugger using the L command when the map file is converted to a symbol file using the MAPSYM utility.

Symbol Absolute. An Absolute symbol is a symbolized constant value that has been made public by the programmer. Such symbolic definitions appear in the map file output from the linkage editor and may be referenced in the Dump Formatter and Kernel Debugger using the LA command when the map file is converted to a symbol file using the MAPSYM utility.

Symbol Group. A symbol group is the set of symbols that are defined within a program segment. Frequently a program segment is given its own selector at load time.

Symbol Map. A symbol map is created from symbolic name information generated by a program compiler and converted for used by the Dump Formatter and Kernel Debugger by the linkage editor and MAPSYM utilities. This allows program code and data locations to be referred to by name as well as by address.

System File Number (SFN). A System File Number (SFN) is the system-wide unique handle by which an open file system object is known. It is the offset into the SFT segment that locates the corresponding SFT entry.

Refer to the following for more related information:

3.4.2, ".A - Format the System Anchor Segment (SAS)" on page 149.

3.4.5, ".D - Display an OS/2 System Structure" on page 160.

Task. A task is a hardware architected thread of execution. The INTEL architecture allows for multiple independent tasks to co-exit and provides the task gate mechanism as a means of switching between tasks. Tasks are represented to the hardware by the TSS.

The characteristics of a task are very similar to that of the OS/2 process. Protect-mode processes however, tend to run under a single task in OS/2 and implement switching through the more efficient software managed context switching mechanism.

Only VDMs and error recovery processes run as independent tasks.

See the *INTEL486 Programmer's Reference* for more information.

Task State Segment (TSS). The Task State Segment (TSS) is a hardware architected control block that is used for two purposes:

1. To implement the privileged level transition mechanism initiated with a Call Gate instruction.
2. To provide a register save area for hardware task switching initiated with a call to a Task Gate.

In general OS/2 does not use the hardware task switching mechanism, so TSSs are few. It does however use the TSS for implementing Application Programming Interfaces (APIs) in the system.

A TSS may be formatted using the Kernel Debugger and Dump Formatter DT command.

Translation Lookaside Buffer (TLB). The Translation Lookaside Buffer (TLB) is a hardware implemented buffer used for caching linear to physical address mappings.

The Intel486(TM) processor provides test registers for manipulating the TLB.

Thrashing. Thrashing refers to the state of a system where most of the CPU time is spent paging in and out memory from the swap file. This happens when real storage is heavily over committed and storage references encompass a wide range of virtual pages over a short processing time.

Such a condition can indicate a poorly tuned application where paging is caused by the process of accessing data the application needs. A typical scenario is where work data is chained in a single, very extended, queue and no mechanism exits to access the required data without scanning the entire chain. Use of hashing techniques greatly reduce this problem.

Thread. A thread is a independently scheduleable entity that competes for processor resource with other threads.

Each thread is represented by a TCB and TSD structure.

Threads are organised within processes and assigned a unique identifier within the owning process known as the Tid.

All threads within the system are assigned a system wide unique identifier known as the Thread Slot Number.

Refer to 3.4.20, ".P - Display Process Status" on page 238 for more detailed information.

Thread Control Block (TCB). The Tread Control Block (TCB) contains per-thread control and status information that must be resident at all times. The swappable counterpart to the TCB is the TSD

Refer to the following for related information

3.4.20, ".P - Display Process Status" on page 238.

Thread Identifier (tid). The Thread Identifier (tid) is a value, unique within the owning processes, used to identify the thread. It is not the same as the Thread Slot Number, which uniquely identifies a thread, system-wide.

The Tid is used in thread related APIs such as DosKillThread and DosSetPriority.

Thread Information Block (TIB). The Thread Information Block (TIB) is a supplemental thread related control block made accessible to ring 3 programs. It contains thread information obtained from the thread's TCB and acts as an anchor for exception-handlers registered for the thread.

The TIB may be located from TCBptib(TCB + 0x10) using the Dump Formatter or Kernel Debugger.

A program gains access to the TIB along with the PIB by calling the DosGetInfoBlocks API.

Thread Local Memory Area (TLMA). The **Thread Local Memory Area (TLMA)** a an area of private arena memory that is instanciated at a thread level. This is achieved by copying the contents of the TLMA to dfff:0024 when a thread switch occurs. The TLMA contents are saved in the TCB at **TCBTLMA**.

Storage is allocated from the TLMA by using the **DosAllocThreadLocalMemory** API.

Thread Slot Number. The Thread Slot Number is a system wide unique identifier assigned to each thread in the system.

Threads are located from the thread slot table whose linear address is at global symbol:

_papTCBSlots

Each slot is a double-word linear address of the corresponding thread's TCB. The first slot (slot=0) is reserved.

Under the Kernel Debugger and Dump Formatter the following symbols may be used to represent particular threads in many of the commands that accept a slot number as a parameter:

***** The current or last dispatched thread as recorded in word global variable **_TaskNumber**

\# The default thread slot used by the Dump Formatter and Kernel Debugger.

Refer to 3.4.20, ".P - Display Process Status" on page 238 for more detailed information.

Thread Swappable Data (TSD). The Thread Swappable Data (TSD) control block contains per-thread status and control information that resides in swappable memory and therefore is not required for reference out of context of the related thread. The resident memory counterpart to the TSD is the TCB (Thread Control Block).

The vast majority of the TSD is used as the ring 0 stack when a thread makes a privilege level transition to ring 0 via a call gate descriptor. The base of the ring 0 stack will therefore include the ring 3 call gate stack frame on entry to ring 0 (which is usually kernel or device driver code).

In the debug kernel a dummy page prefixes the used part of the TSD in order to catch ring 0 stack faults.

Other information contained in the TSD includes GTD instance data for the corresponding thread's context. This comprises descriptors for:

28: The LDT descriptor.

30: Base selector for ring 0 process instance data, which includes the ring 0 stack, TCBs and PTDA.

38: Floating point emulator instance data

40: FS mapping to the TIB

When an an inter-process thread context switches, descriptors 30 - 40 are loaded into the GDT from the TSD. When an intra-process thread context switches, descriptors 28 - 40 are loaded into the GDT from the TSD.

Refer to the following for related information

3.4.20, ".P - Display Process Status" on page 238.

Thunking. Thunking is the process of calling 16-bit code from 32-bit code and vice versa. Thunking consists of applying the CRMA to convert from one form of address to the other and making any stack parameter adjustments either by padding 16-bit operands to 32-bit with leading zeros (16- to 31-bit conversion) or truncating the padded 32-bit value to 16 bits (32- to 16-bit conversion).

Tracepoint. A **tracepoint** is designated location in system or application code where the System Trace Facility will gather data for logging by the STDA.

Tracepoints may be implemented statically by use of the **DosSysTrace** API or dynamically through use of the Dynamic Trace Customizer.

System defined tracepoints are documented in the *System Tracepoints Reference.*

UVIRT. The UVIRT attribute signifies virtual storage mapping to a pages of physical storage.

The full set of memory management structures associated with virtual storage allocation may not exist for UVIRT storage.

The UVIRT attribute may be associated with a number of structures, for example:

PTE

LDT and GDT descriptors

VMAL

In general UVIRT allocations are 'convenience' mappings memory to selectors. Typically they are created by device drivers using the DevHlp_PhysToUvirt facility.

Virtual DOS Machine (VDM). A Virtual DOS Machine (VDM) is a type of process that runs in an emulated DOS environment using the DOS EMulation (DEM) component of OS/2.

Virtual Page Structure (VP). A Virtual Page Structure (VP) is used by memory management to track the status of a virtual storage frame, whether backed by physical storage, cached by the loader or paged out to the swapper. The Virtual Page Structures are allocated in contiguous storage, anchored from the address specified in global variable:

_pgpVPBase

Refer to the following for more detailed information

3.4.18, ".MV - Format Memory Virtual Page Structures (VPs)" on page 230.

Virtual Memory Arena Header Record (VMAH). One Virtual Memory Arena Header Record (VMAH) is allocated per arena to record information about the address range of an arena. The VMAH points to its sentinel arena record (VMAR).

Each VMAH chained in a double linked list.

The system arena VMAH is located at global symbol:

_ahvmSys

The shared arena VMAH is located at global symbol: _ahvmShr

For each private arena the VMAH is imbedded in the PTDA at label &ptdaah..

Under OS/2 2.1 the system and shared arena VMAHs are assigned to objects 4 and 5 respectively.

Virtual Memory Alias Record (VMAL). The Virtual Memory Alias Record (VMAL) is used to represent aliased regions of virtual memory. These are either:

regions of physical storage that may be addressed by more than one virtual or

linear address that are not associated with a memory object, such as VDM UVIRT allocations.

When two memory objects are aliases of each other then they need not have co-incident sizes or origins within the aliased arena record. Aliases are designed to provide alternative attributes for accessing the same piece of data within or across processes. Compare this with shared instance data within the shared arena, where multiple object records share a common arena record. In this case each object is associated with a unique process and is not considered an alias.

Each VMAL is identified by a unique handle referred to as the hal.

Refer to the following for more detailed information

3.4.15, ".ML - Format Memory Alias Records (VMAL)" on page 209.

Virtual Memory Kernel Heap (VMKH). The Virtual Memory Kernel Heap (VMKH) structures are used to describe system heap memory. Many objects allocated out of the kernel heap are assigned a System Object identifier.

Virtual Memory Arena Record (VMAR). The Virtual Memory Arena Record (VMAR) is used to represent a contiguous region of virtual memory allocated in page quantities. Such storage may or may not be committed or resident.

Arena records are chained in a doubly linked lists, one for each arena type. That is, the chain chain exists separately for each private arena, the shared arena and system arena.

Special arena records, known as Sentinels head each chain. They describe the entire arena which they head.

All virtual memory is described by by at least one arena record.

Each VMAR is identified by a unique handle referred to as the har.

Arena also records point to the following related memory structures:

VMOB

VMAL

VMCO

Refer to the following for more detailed information

3.4.15, ".ML - Format Memory Alias Records (VMAL)" on page 209.

Virtual Memory Context Record (VMCO). A Virtual Memory Context Record (VMCO) is used to record the association of shared arena, shared data objects with processes that are using.

Each VMCO is identified by a unique handle referred to as the hco.

Refer to the following for more detailed information

3.4.13, ".MC - Format Memory Context Records (VMCO)" on page 204.

Virtual Memory Object Record (VMOB). The Virtual Memory Object Record (VMOB) are used to represent memory objects, that is the instance data associated with a particular virtual address. VMOBs contain pointers to the the owning and requesting objects as well as the corresponding arena record (VMAH).

Each VMOB is identified by a unique handle referred to as the hob.

Refer to the following for more detailed information

3.4.16, ".MO - Format Memory Object Records (VMOB)" on page 212.

Volume Parameter Block (VPB). A Volume Parameter Block (VPB) is used to store volume information associated with a file system object. All VPBs are contained within a single segment locatable from the SAS. Most file system structures contain a VPB handle for an associated volume. The handle is used as an offset into the VPB segment.

See also related structures:

CDS

MFT

SFT

DBP

FSC

Refer to the following for more detailed information

3.4.2, ".A - Format the System Anchor Segment (SAS)" on page 149

3.4.5, ".D - Display an OS/2 System Structure" on page 160

Zombie. The term **Zombie** is used to describe a.Z terminal condition of a thread or process. There is a strict operating system definition and two colloquial uses:

-
- The strict system definition refers to a process that has terminated but whose PTDA has been retained on the zombie queue (**_pPTDAFirstZombie**) because the process status byte (LISEG+0xa) indicates that its parent wishes to collect termination information through **DosWaitChild**. The dead child is retained on the zombie queue until either the parent dies or issues **DosWaitChild**.

- Zombie is also commonly used to refer to a terminating thread or process that has blocked after the application has returned to the operating system. Usually this implies a problem freeing memory because one or more pages have been long-term locked by a device driver.
- The third use of zombie refers to any process that is anonymous. Internal thread, VDMs, and terminating threads can be anonymous.

List of Abbreviations

AAB	Application Anchor Block
BMP	Block Management Package
CDA	Common ABIOS Data Area
CDIB	Codepage Data Information Block
CDS	Current Directory Structure
CRI	Client Register Information
DEM	DOS Emulation
DPB	Driver Parameter Block
FAT	File Allocation Table
FSC	File System Control Block
FSD	File System Driver
GDT	Global Descriptor Table
GISEG	Global Information Segment
IBM	International Business Machines Corporation
IDT	Interrupt Descriptor Table
IPE	Internal Processing Errors
ITSO	International Technical Support Organization
JFN	Job File Number
JFT	Job File Number Table
LDT	Local Descriptor Table
MQ	Message Queue Header
MFT	Master File Table
MTE	Module Table Entry
OTE	Object Table Entry
PAI	Physical Arena Information block
PF	Page Frame Structure
PIB	Process Information Block
Pid	Process Identifier
PSP	Program Segment Prefix
PTDA	Per-Task Data Area
PTE	Page Table Entry
PTREE	Patricia Tree
QMSG	Queue Message
RAS	Reliability, Availability and Serviceability
RIP	Register Information Packet

RLR	Record Lock Record
RMP	Record Management Package
SAS	System Anchor Segment
SDTA	System Trace Data Area
SFN	System File Number
SFT	System File Table
SGCB	Screen Group Control Block
SMS	Send Message Structure
SMTE	Swappable Module Table Entry
SQMSG	System Queue Message
STE	Segment Table Entry
TCB	Thread Control Block
TIB	Thread Information Block
Tid	Thread Identifier
TLMA	Thread Local Memory Area
TLB	Translation Lookaside Buffer
TLMA	Thread Local Memory Area
TSD	Thread Swappable Data
TSS	Task State Segment
VDM	Virtual DOS Machine
VMAH	Virtual Memory Arena Header Record
VMAL	Virtual Memory Alias Record
VMAR	Virtual Memory Arena Record
VMCO	Virtual Memory Context Record
VMKH	Virtual Memory Kernel Heap
VMOB	Virtual Memory Object Record
VP	Virtual Page Structure
VPB	Volume Parameter Block
WND	Window Sturcture

Index

Special Characters

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z

ITSO Technical Bulletin Evaluation RED000

International Technical Support Organization
OS/2 Debugging Handbook - Volume II
Using the Debug Kernel and Dump Formatter
February 1996

Publication No. SG24-4641-00

Your feedback is very important to help us maintain the quality of ITSO Bulletins. **Please fill out this questionnaire and return it using one of the following methods:**

- Mail it to the address on the back (postage paid in U.S. only)
- Give it to an IBM marketing representative for mailing
- Fax it to: Your International Access Code + 1 914 432 8246
- Send a note to REDBOOK@VNET.IBM.COM

Please rate on a scale of 1 to 5 the subjects below.
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction	____		
Organization of the book	____	Grammar/punctuation/spelling	____
Accuracy of the information	____	Ease of reading and understanding	____
Relevance of the information	____	Ease of finding information	____
Completeness of the information	____	Level of technical detail	____
Value of illustrations	____	Print quality	____

Please answer the following questions:

a) If you are an employee of IBM or its subsidiaries:

Do you provide billable services for 20% or more of your time? Yes____ No____

Are you in a Services Organization? Yes____ No____

b) Are you working in the USA? Yes____ No____

c) Was the Bulletin published in time for your needs? Yes____ No____

d) Did this Bulletin meet your needs? Yes____ No____

If no, please explain:

_____

_____

What other topics would you like to see in this Bulletin?

_____

_____

What other Technical Bulletins would you like to see published?

_____

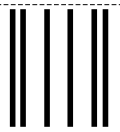
Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

_____ _____
Name Address

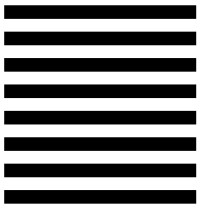
_____ _____
Company or Organization

_____ _____
Phone No.

Fold and Tape **Please do not staple** Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM International Technical Support Organization
Department 626C, Building 014-1
Internal Zip 5220
1000 Northwest 51st Street
Boca Raton, Florida
USA 33431-1328

Fold and Tape **Please do not staple** Fold and Tape

Printed in U.S.A.